Apple® II

Einführung in Applesoft BASIC



Apple IIc, Apple IIe, Apple IIGS.

20525 Mariani Ave.
Cupertino, CA 95014
(408) 996-1010

Apple Computer GmbH
Ingolstädter Str.20
D-8000 München 45
(089)-350340

Apple® II Einführung in Applesoft BASIC

Inhalt

Vorwort

In diesem Tutorium werden Sie lernen, einfache Applesoft BASIC Computerprogramme mit Ihrem Apple® II Computer zu schreiben. Allein aus diesem Tutorium werden Sie allerdings nicht alles über Applesoft BASIC lernen. Sobald Sie jedoch die zehn Lektionen durchgearbeitet haben, können Sie entscheiden, ob Sie mehr über das Programmieren erfahren möchten oder nicht.

Die Trainings-Diskette, die mit Ihrem Computer geliefert wurde, enthält eine kurze Einführung in Applesoft. Möglicherweise sollten Sie etwas mit dieser Diskette arbeiten, bevor Sie dieses Tutorium lesen.

Was ist eine Computersprache?

Eine Computersprache ist wie eine Sprache, die die Menschen sprechen. Sie hat ein Vokabular und eine Syntax - die Reihenfolge der Wörter und die Schreibweise sind wichtig. Ihr Apple Computer spricht eine Sprache namens **Applesoft BASIC**. (Er spricht auch noch andere Sprachen, sie sind jedoch nicht in den Computer eingebaut; sie können auf Disketten erworben werden.) Der Computer liest die über die Tastatur eingegebenen BASIC-Befehle und führt die Instruktionen genau aus. Glücklicherweise kann BASIC einfacher erlernt werden als eine menschliche Sprache, da BASIC über weniger Wörter verfügt und seine Grammatik im allgemeinen sehr geradlinig ist.

❖ *Übrigens...:* Es gibt viele Variationen der BASIC-Computersprache. In diesem kleinen Tutorium beziehen sich jedoch die Ausdrücke *BASIC*, *Applesoft BASIC* und *Applesoft* alle auf dieselbe Sache.

Was ist ein Programm?

Bei der Programmierung werden Instruktionen für den Computer geschrieben. Die ganzen Instruktionen, die an einen Computer gesendet werden, damit er eine bestimmte Funktion ausführt, stellen das **Programm** dar. Angenommen, Ihr Computer ist ein kleiner Hund, der dressiert werden soll. Sie können mit Ihrem Hund wie mit einem Menschen sprechen. Mit einem begrenzten Vokabular wird ihm genau gesagt, was er tun muß. Soll er eine Reihe von Dingen ausführen, so können Sie ihm eine Reihe von Befehlen geben, einen nach dem anderen. Angenommen, Sie möchten, daß Ihr Hund sitzt, sich legt und sich umdreht. In diesem Fall wird folgendermaßen vorgegangen:

```
"Waldi, sitz."
(Waldi setzt sich.)
"Waldi, lieg."
(Waldi legt sich.)
"Waldi, dreh dich um."
(Waldi dreht sich um.)
"Guter Hund!"
(Waldi wedelt mit dem Schwanz.)
```

Natürlich setzt sich Ihr Apple Computer nicht, legt sich nicht hin oder dreht sich um. Allerdings tut er eine ganze Reihe von Dingen für Sie, wenn Sie ihm die Instruktionen in einer systematischen und logischen Reihenfolge geben. Bei der Computer-Programmierung wird dieselbe Direktheit, Einfachheit und Reihenfolge benutzt, wie beim Dressieren eines Hundes (allerdings brauchen Sie den Computer nicht zu loben, wenn er das tut, was Sie ihm auftragen).

Müssen Sie programmieren?

Sie brauchen keine Programme zu schreiben, um Ihren Computer benutzen zu können. Tausende von Programmen wurden schon für Ihren Apple geschrieben - Programme für die Textverarbeitung, Finanzanalyse, computergestützte Ablage und Dutzende von anderen Anwendungen. Sie legen einfach eine Diskette mit den entsprechenden Programmen in das Diskettenlaufwerk und schalten den Computer ein.

Warum wird programmiert?

In erster Linie werden Sie viel Spaß beim Programmieren haben. Beim Programmieren werden Sie feststellen, daß Ihr Apple keine Wunderdinge vollbringt (obwohl es gelegentlich so aussieht); er befolgt einfach die Instruktionen, die Sie ihm geben. Durch das Programmieren führt der Computer einfach die Dinge aus, die *Sie* haben möchten - es sind also alles Ihre eigenen Wunderdinge. Zweitens lernen Sie viel über die Arbeitsweise eines Computers, während Sie lernen, ihn zu programmieren. Dadurch erhalten Sie ein besseres Verständnis der Dinge, die der Computer ausführen und die er nicht ausführen kann. Schließlich werden Sie feststellen, daß das Programmieren eine faszinierende Sache ist und Ihre eigene Kreativität auf eine Art und Weise anregt, von der Sie nie geträumt haben. Vielleicht möchten Sie das Programmieren sogar zu Ihrem Beruf machen.

Sie können einfache Unterhaltungs-, Ausbildungs- und kommerzielle Programme mit einem Grundvorrat an Instruktionen erstellen. So können Sie beispielsweise sehr gute Spieleprogramme mit Ausbildungseffekt oder private Ausgabenprogramme in Applesoft BASIC schreiben, um Ihre Finanzen in Ordnung zu halten.

Das Schreiben eigener Programme ist eine *Möglichkeit*, die bei Ihrem Apple zur Verfügung steht. Wenn Sie auch wahrscheinlich feststellen werden, daß das Programmieren nützlich und interessant ist, so brauchen Sie nicht zu wissen, wie man programmiert, um den Computer zu benutzen. *Möchten* Sie jedoch programmieren, so werden Sie feststellen, daß Appesoft BASIC hierfür ein ausgezeichneter Ausgangspunkt ist.

Bitte haben Sie Geduld

Wie alles, so braucht auch das Programmieren etwas Geduld. Sie müssen ein erfahrener Koch sein, um ein erstklassiges Sieben-Gänge-Menü zu kochen. Die Grundlagen beginnen jedoch mit dem Erhitzen von Butter, dem Aufschlagen von Eiern usw. Dasselbe gilt für das Ergebnis. Sie brauchen kein Meisterkoch zu sein, um ein Omelett zu machen (oder Ihre Freunde mit Ihren Kochkenntnissen zu überraschen).
Gelegentlich brauchen Sie ganz einfach Geduld - und das auch nur am Anfang. Sie dürfen nur den Glauben nicht verlieren.

Die ersten Schritte

Applesoft ist in Ihren Apple II Computer eingebaut. Sie müssen jedoch den Computer so vorbereiten, daß die erstellten Programme gespeichert werden, damit sie erneut benutzt werden können. (In Lektion 4 wird das Speichern von Programmen auf Diskette im einzelnen erläutert.) Hier die ersten Schritte mit Applesoft BASIC:

1. Lesen Sie als erstes das Benutzerhandbuch. Es enthält wichtige Informationen über den Computer, die Sie kennen müssen, bevor Sie mit Applesoft arbeiten können. Beachten Sie insbesondere den Abschnitt über das Formatieren von Disketten. Sie benötigen mindestens eine formatierte Diskette, bevor Sie beginnen können.

2. Legen Sie die mit Ihrem Computer gelieferte Dienstprogramm-Diskette in das Laufwerk, schließen Sie die Diskettenverriegelung und schalten Sie den Computer ein. (Dies wird im Benutzerhandbuch beschrieben.) Wählen Sie die Option Applesoft BASIC und drücken Sie Return; darauf wird folgendes Symbol angezeigt:].

3. Nehmen Sie die Diskette aus dem Laufwerk heraus und ersetzen Sie sie durch eine formatierte Diskette. Die Diskettenverriegelung muß unbedingt geschlossen werden.

❖ *Benutzung von Applesoft ohne Diskettenlaufwerk:* Auch ohne Diskettenlaufwerk können Programme geschrieben werden. Allerdings können sie nicht gespeichert werden. Um BASIC ohne ein Diskettenlaufwerk zu starten, schalten Sie den Computer ein und drücken die Control- und Reset-Tasten gleichzeitig. Danach lassen Sie sie wieder los. Folgendes Symbol wird angezeigt:].

Und nun können wir beginnen!

Dieses Tutorium ist in zehn Lektionen unterteilt: für jede Lektion brauchen Sie etwa eine Stunde. Sie sollten sich unbedingt genügend Zeit nehmen, um die in jeder Lektion erlernten Dinge zu üben, bevor Sie zur nächsten Lektion weitergehen. Jede Lektion baut auf der vorhergehenden Lektion auf.

Vor allem wünschen wir Ihnen viel Spaß. Experimentieren Sie so viel wie nur möglich. Brechen Sie die Regeln. Versuchen Sie verrückte Dinge - das schlimmste was passieren kann, ist ein akustisches Signal vom Computer. (Geben Sie es einfach zurück.) Das Programmieren soll in erster Linie Spaß machen. Nun blättern Sie einfach weiter und fangen an.

Lektion 1

Die ersten Schritte

Ob Sie gerne programmieren, finden Sie am besten heraus, indem Sie ein wenig üben. Um die Dinge problemlos zu gestalten, führen Sie die Schritte einfach so aus, wie sie in diesem Tutorium dargelegt werden. Sollten Sie sich langweilen, so können Sie natürlich eigene Experimente wagen! Der Computer kann nicht kaputtgehen, wenn Sie eine falsche Eingabe machen. Am wichtigsten ist es, daß Sie experimentieren, lernen und dabei noch Spaß haben.

In dieser ersten Lektion werden die Grundbegriffe erläutert. Hier werden Programmzeilen und Zeilennummern beschrieben. Außerdem wird erläutert, wie Programme eingegeben werden. Sie werden sehen, wie Meldungen mit der PRINT-Instruktion auf den Bildschirm gesetzt werden. Außerdem werden Sie einiges über Programmierfehler und deren Behebung erfahren.

Grundlagen

Bevor Sie irgendetwas anderes tun, geben Sie das Wort NEW ein und drücken die Return-Taste. Durch **NEW** weiß Ihr Apple Computer, daß ein neues Programm eingegeben werden soll. Durch Return betrachtet der Apple die gerade vorgenommene Eingabe. Bis Return gedrückt wird, glaubt Ihr Apple einfach, daß Sie mit sich selbst reden!

```
NEW
```
Hier Return drücken.

Nun geben Sie die folgende Zeile genau wie dargestellt ein und drücken danach Return:

```
10 PRINT "SITZ"
```
Hier Return drücken.

Die Zahl 10 wird als **Zeilennummer** bezeichnet. Ihr Apple führt die eingegebenen Instruktionszeilen in numerischer Reihenfolge aus, wobei immer mit der niedrigsten Nummer begonnen wird. Am Anfang sollten die Programmzeilen immer in Abständen von 10 numeriert werden. In Lektion 3 wird beschrieben warum.

Nachdem sämtliche Instruktionen eingegeben wurden (dies ist hiermit geschehen - das erste Programm ist nur ein kurzes Programm), geben Sie RUN ein und drücken Return. Durch den **RUN**-Befehl weiß Ihr Apple, daß die Instruktionen beendet sind und nun ausgeführt werden sollen:

```
RUN
```
Hier Return drücken.

Die Bildschirmanzeige müßte nun folgendermaßen aussehen:

```
]NEW

]10 PRINT "SITZ"

]RUN

SITZ

]▌
```

Damit haben Sie nun Ihr erstes Computer-Programm geschrieben und **ausgeführt**. Gut gemacht! Außerdem haben Sie einen der am häufigsten benutzten Programmierbefehle kennengelernt: PRINT. Durch die PRINT-Instruktion weiß der Computer, daß er die in Anführungszeichen stehenden Angaben anzeigen muß. Hier noch einige weitere Übungen mit PRINT. Geben Sie das folgende Programm genau wie angegeben ein. (Machen Sie einen Fehler, so drücken Sie einfach Return und geben die Zeile erneut ein.) Die Return-Taste muß unbedingt am Ende jeder Zeile gedrückt werden:

```
10 print "lieg"
20 Print "Dreh dich um"
30 pRiNt "SteH aUf"
RUN
```

Auf dem Bildschirm steht folgendes:

```
lieg
Dreh dich um
SteH aUf
```

❖ *Warum wird NEW hier nicht benötigt?* Wird eine Zeilennummer erneut benutzt, so ersetzt die neue Zeile die alte Zeile. Das als letztes eingegebene Programm umfaßte nur eine Zeile - Zeile 10. Dieses neue Programm verfügt ebenfalls über eine Zeile 10, die die alte Zeile ersetzt. Dies ist genau so, als hätten Sie NEW eingegeben.

Für den Computer spielt es keine Rolle, ob die Buchstaben als Groß- oder Kleinbuchstaben oder als Kombination der beiden eingegeben werden. Allerdings müssen Sie darauf achten, wie Sie Ihre Instruktionen eingeben. Der Computer erwartet, daß Sie ihm auf eine *für ihn verständliche Weise* genau sagen, was er tun muß. Ansonsten erhalten Sie eine Fehlermeldung wie diese:

```
?SYNTAX ERROR IN 10
```

Computer führen immer *genau* aus, was ihnen gesagt wird, und nicht unbedingt das, was Sie ihnen *sagen wollten*. Selbst kleine Tippfehler können zu der Fehlermeldung "Syntaxfehler" führen. (Im allgemeinen wird eine Zeilennummer angegeben, damit Sie den Fehler finden können.) Geben Sie

```
NEW
```

ein und drücken Sie Return. Danach geben Sie dieses eine Zeile umfassende Programm ein und versuchen, es auszuführen:

```
10 PRIMT "HOPPLA"
```

(Vergessen Sie Return am Ende der Zeile nicht - wir werden Sie nicht noch einmal darauf aufmerksam machen.)

Nachdem das Programm ausgeführt werden sollte, wird folgende Fehlermeldung angezeigt:

```
?SYNTAX ERROR IN 10
```

10 ist die Zeilennummer.

Auch wenn Sie und jeder andere Benutzer wissen, daß Sie PRINT anstelle von PRIMT *sagen wollten*, hat diese Instruktion den Apple verwirrt. Glücklicherweise zeigt der Computer bei den meisten Fehlern eine eingebaute Fehlermeldung an, so daß Sie wissen, was Sie falsch gemacht haben. Je mehr Sie programmieren (und natürlich je mehr Fehler Sie machen), desto mehr Fehlermeldungen werden angezeigt, anhand derer Sie verstehen können, wie der Computer arbeitet. *Denken Sie daran:* der Computer zeigt Fehlermeldungen an, damit Sie Fehler korrigieren können, und nicht um Ihnen zu sagen, daß Sie dumm sind. Betrachten Sie diese Fehlermeldungen als hilfreich und nicht als störende Begleiterscheinungen.

Editieren: Erste Hilfe beim Programmieren

Wie Sie gerade gesehen haben, müssen Sie bei der Eingabe eines Computerprogramms aufpassen, um die Eingabe eines Fehlers zu vermeiden. Viele Fehler sind einfach das Ergebnis von Tippfehlern. Sie können sich viel Mühe sparen, indem Sie Ihre Schreibweise während der Eingabe prüfen.

Es wird sehr schnell mühsam, eine Zeile immer wieder neu einzugeben, nachdem Sie einen einfachen Tippfehler gemacht haben. Ihr Apple verfügt über einige eingebaute Einrichtungen, mit denen solche Fehler einfacher korrigiert werden können.

Geben Sie die folgende Zeile ein, drücken Sie jedoch noch nicht Return:

```
10 PRINT K "ACHTUNG, EIN FEHLER"
```

noch nicht Return drücken!

Das *K* zwischen der PRINT-Instruktion und der Meldung wird zu Problemen führen. Nun *könnten* Sie die ganze Zeile neu eingeben. Muß dies jedoch bei jedem Fehler geschehen, so kommen Sie nie weiter. Statt dessen suchen Sie nach den vier Pfeiltasten in der unteren rechten Ecke der Tastatur. Danach tun Sie folgendes:

1. Drücken Sie die Linkspfeil-Taste, bis der Cursor direkt über dem störenden K steht.
2. Drücken Sie die Leertaste einmal, um das K zu löschen (benutzen Sie nicht die Delete-Taste; sie kann mit Applesoft nicht benutzt werden).
3. Mit der Pfeiltaste nach rechts bewegen Sie den Cursor, bis er rechts neben dem letzten Anführungszeichen in der Zeile steht. (Drücken Sie Return in der Mitte der Zeile, so sind alle Angaben ab dieser Stelle bis zum Zeilenende verloren.)
4. Nun prüfen Sie, ob die Zeile richtig ist.

Sie müßte folgendermaßen aussehen:

```
10 PRINT "ACHTUNG, EIN FEHLER"
```

Nun kann Return gedrückt und das Programm ausgeführt werden.

Zusammenfassung und Überblick

In dieser ersten Lektion haben Sie gelernt, wie der Weg für neue Programme mit NEW freigegeben wird, wie Programme mit RUN ausgeführt werden, und wie Meldungen mit PRINT auf den Bildschirm gesetzt werden. Sie haben gesehen, wie die Instruktionsfolge mit Zeilennummern in dem Programm angeordnet wird. Schließlich haben Sie etwas über Fehler und ihre Beseitigung erfahren.

Bevor Sie nun zu der nächsten Lektion gehen, üben Sie ein wenig mit der PRINT-Instruktion. Schreiben Sie ein fünf Zeilen umfassendes Programm. Danach ändern Sie die Zeilennummern, indem Sie die Zeilen neu eingeben (machen Sie beispielsweise die letzte Zeile zur ersten), um zu sehen, was geschieht. Und keine Angst vor Fehlern - es gibt keine Noten.

Lektion 2

Arithmetik und Variablen

Sie brauchen nicht viel von der Arithmetik zu verstehen, um Ihren Apple Computer programmieren zu können. Die meisten Programme benötigen jedoch arithmetische Funktionen, um die gewünschten Aufgaben ausführen zu können. (So muß beispielsweise in einem Programm für die Verfolgung ausgegebener Schecks der Betrag jedes Schecks vom Konto abgezogen werden.) In dieser Lektion werden die Grundlagen der Computer-Arithmetik erläutert. Außerdem erhalten Sie Informationen über Variablen, die Bereiche im Speicher des Computers, in denen Werte stehen. Schließlich wird beschrieben, wie Variablen Namen zugewiesen werden, damit sie einfacher benutzt werden können - danach werden wir Sie auffordern, diese Regeln zu brechen, um zu sehen, was geschieht.

Arithmetik

In der ersten Lektion wurde beschrieben, daß Ihr Apple sämtliche in Anführungszeichen hinter der PRINT-Instruktion stehenden Angaben anzeigt. Zur Ausführung arithmetischer Operationen wird die PRINT-Instruktion *ohne* Anführungszeichen benutzt.

Geben Sie beispielsweise folgendes Programm ein und führen Sie es aus:

```
NEW
10 PRINT "5 + 5"
20 PRINT 5 + 5
RUN
5 + 5
10
```

`5 + 5`	In Zeile 10 wurde genau ausgedruckt, was in den Anführungszeichen steht.
`10`	In Zeile 20 wurde die Summe der beiden Zahlen ausgedruckt.

Mit der ersten Zeile haben Sie Ihren Apple angewiesen, den Satz "5 + 5" auszudrucken. In der zweiten Zeile haben Sie jedoch gesagt "Addiere die Zahlen 5 + 5 und zeige das Ergebnis an."

Natürlich kann Ihr Apple noch wesentlich mehr als nur addieren. Er kann sogar einige sehr komplexe mathematische Aufgaben ausführen. In diesem Tutorium wollen wir jedoch bei den Grundoperationen bleiben: Addition, Subtraktion, Multiplikation und Division. Nachfolgend eine Tabelle mit den Symbolen (die als Operatoren bezeichnet werden), die der Computer zur Ausführung einfacher arithmetischer Aufgaben benutzt:

Operator	Aktion
+	Addition
-	Subtraktion
*	Multiplikation
/	Division

Die Operatoren für Addition und Subtraktion bleiben die gleichen wie immer. Wahrscheinlich kennen Sie den Operator für die Division auch schon, mit dem ein Bruch ausgedrückt wird (wie beispielsweise in 7/8). Nur der Operator für die Multiplikation sieht etwas anders aus. Hier handelt es sich um ein Sternchen (*) anstelle eines X. Viele Programmierer benutzen den Buchstaben X zur Darstellung eines unbekannten Wertes, so daß eines Tages entschieden wurde, für die Multiplikation statt dessen das Sternchen zu benutzen (das wie ein X aussieht, nur einfach mit einem zusätzlichen Strich in der Mitte).

Nachfolgend ein Beispielprogramm. Geben Sie es ein; bevor Sie es ausführen, überlegen Sie jedoch, wie die Ergebnisse aussehen werden:

`10 PRINT 4 + 5`	Eine einfache Addition.
`20 PRINT 7.56 - 4.44`	Mit Dezimalstellen hat Ihr Computer keine Probleme.
`30 PRINT 4 * 5`	Denken Sie daran: * bedeutet Multiplikation.
`40 PRINT 4.6 / 2`	Hier eine einfache Division.
`50 PRINT 11 + 12 - 13 + 14`	Er kann auch mehrere Operationen ausführen.
`60 PRINT 12 / 3 + 4`	Der Computer löst Aufgaben von links nach rechts ...
`70 PRINT 10 * 2 + 8 / 2`	... hier müssen jedoch noch einige Punkte berücksichtigt werden (lesen Sie die Angaben über die Priorität im nächsten Abschnitt).

Aus Zeile 20 ersehen Sie, daß Ihr Computer Brüche verarbeiten kann - sie müssen nur einfach so ausgedrückt werden, daß er sie verstehen kann. Möchten Sie dem Computer beispielsweise sagen, daß er die Summe von 2 1/2 + 3 errechnen soll und geben Sie folgendes ein:

```
PRINT 2 1/2 + 3
```

so erhalten Sie eine Antwort, mit der Sie nicht gerechnet haben. Der Computer zeigt 13,5 anstatt 5,5 an. Er interpretiert 2 1/2 + 3 als "dividiere die Zahl 21 durch 2 und addiere 3 zu dem Ergebnis". Leerstellen zwischen den Zahlen haben für Ihren elektronischen Freund keine Bedeutung.

Haben Sie alle Aufgaben vor Ausführung des Programms im Kopf durchgerechnet, so ist die letzte Antwort vielleicht eine Überraschung:

`70 PRINT 10 * 2 + 8 / 2`	Die Antwort ist 24 und nicht 14.

Das Ergebnis der Berechnungen ist auf die **Priorität** zurückzuführen. Mit Priorität wird die Reihenfolge angegeben, in der der Computer mathematische Operationen ausführt.

Priorität: Reihenfolge der Berechnungen

Im allgemeinen rechnet Ihr Apple von links nach rechts. Allerdings werden sämtliche Multiplikationen und Divisionen vor der Addition und Subtraktion ausgeführt. Führen Sie die Berechnungen in Zeile 70 aus, um zu sehen, wie sich die Priorität auswirkt.

Rechnung:	10 * 2 + 8 / 2
Schritt 1:	10 * 2 = 20
Schritt 2:	8 / 2 = 4
Schritt 3:	20 + 4 = 24

Benutzung von Klammern zur Änderung der Priorität

Gelegentlich muß die Priorität neu festgelegt werden, so daß zuerst Addition und Subtraktion und erst danach Multiplikation und Division ausgeführt werden. Was geschieht beispielsweise bei

```
PRINT 18 + 4 / 2
```

wenn zuerst 18 und 4 addiert und die Summe danach durch 2 dividiert werden soll? Dies kann mit dem folgenden kleinen Programm gelöst werden:

```
NEW
10 PRINT 18 + 4 / 2              Dies ergibt 20 ...
20 PRINT (18 + 4) / 2            ... dies ergibt jedoch 11.
```

In Zeile 10 wird zuerst die Division ausgeführt und danach das Ergebnis zu 18 addiert. In Zeile 20 wird die Priorität neu festgelegt, indem die Summe in Klammern gesetzt wird. Durch Klammern wird die Prioritätsfolge geändert. Die Angaben in Klammern werden zuerst berechnet, auch hier von links nach rechts und Multiplikation/Division vor Addition/Subtraktion.

Gegebenenfalls können sogar Klammern in andere Klammern gesetzt werden, um die Priorität in noch komplexeren Situationen anzugeben. Es muß daran gedacht werden, daß von dem innersten Klammernpaar nach außen gegangen wird.

Betrachten Sie das nächste Programm und prüfen Sie, ob Sie die Ergebnisse berechnen können, bevor Sie es ausführen:

```
10 PRINT (7-3) * 2
20 PRINT 3 * ((10 - 6) / 2)
30 PRINT ((4  - 3) / (9 + 2)) * 2
40 PRINT (((1 + 2) * (2-1)) + 11) / 10
```

Nun führen Sie das Programm aus und sehen, ob Sie recht hatten.

Wann immer Sie mit vielen Klammern arbeiten, prüfen Sie, ob die Anzahl von linken Klammern gleich der Anzahl von rechten Klammern ist. Stimmen die Klammernpaare nicht überein, so erhalten Sie eine Meldung über einen Syntaxfehler.

❖ *Angenommen, Sie sind der Computer:* Wann immer Sie ein Programm oder einen Teil eines Programms schreiben, führen Sie es im Kopf aus, bevor Sie es mit dem Computer ausführen. Je mehr Sie "Computer spielen", desto schneller verstehen Sie, wie der Computer arbeitet. Auf diese Weise geben Sie die Instruktionen automatisch so ein, wie der Computer sie benötigt; und schon bald erhalten Sie immer weniger Fehlermeldungen. Versuchen Sie es eine Zeitlang und sehen Sie, was geschieht.

Experimentieren Sie mit eigenen Arithmetik-Programmen. Versuchen Sie, die Prioritäten zu mischen. Mischen Sie einige Sätze ein, mit denen beschrieben wird, was Sie tun. Zum Beispiel:

```
NEW
10 PRINT "12 + 20, dividiert durch die Differenz zwischen 5 und 3,5, ergibt "
20 PRINT (12 + 20) / (5 - 3.5)
```

❖ *Überlaufende Zeilen:* Zeigt Ihr Computer 40 Zeichen pro Zeile auf dem Bildschirm an, so kann die Rechnung in Zeile 10 über den Rand des Bildschirms laufen und an den Anfang der nächsten Zeile gesetzt werden. Dabei wurde das Wort *Differenz* geteilt. Mit etwas mehr Erfahrung lernen Sie Tricks, um wenig schön getrennte Wörter zu vermeiden. Bis dahin ignorieren Sie sie einfach - genau wie der Computer.

Damit wissen Sie nun, wie Ihr Apple arithmetische Operationen ausführt. Sie können ihn wie einen Taschenrechner benutzen (obwohl die Benutzung eines Taschenrechners wahrscheinlich schneller und einfacher ist). Die gerade erlernten einfachen arithmetischen Funktionen werden jedoch wesentlich leistungsfähiger, wenn sie mit Variablen benutzt werden.

Variablen

Variablen sind Symbole für Werte. Sie werden als *Variablen* bezeichnet, da sich ihre Werte ändern oder *variieren* können. Variablen sehen wie Ausdrücke aus, die versehentlich nicht in Anführungszeichen gesetzt wurden:

```
NEW
10 PRINT "HALLO"
20 PRINT HALLO
RUN
HALLO                      In Zeile 10 wird dies ausgedruckt.
0                          Das Ergebnis von Zeile 20.
```

In diesem Programm ist das erste HALLO für den Computer ein Ausdruck, der unverändert ausgedruckt wird. Das zweite HALLO ist eine Variable, deren Wert zufällig gleich Null ist. Einer Variablen wird mit dem Gleichheitszeichen (=) ein Wert zugewiesen.

Nun werden folgende Zeilen zu dem HALLO-Programm hinzugefügt und ausgeführt:

```
30 HALLO = 128
40 PRINT HALLO          Dies ergibt 128!
RUN
HALLO
0
128                     Der neue Wert, der der Variablen  HALLO in Zeile 30 zugewiesen wurde.
```

Damit haben Sie gerade einer Variablen namens HALLO den Wert 128 zugewiesen. Stellen Sie sich eine Variable wie ein temporäres Ablagefach vor. Was immer in das Fach gelegt wird, bleibt dort, bis es durch etwas anderes ersetzt wird. Fügen Sie die beiden folgenden Zeilen zu dem Programm hinzu und führen Sie es erneut aus:

```
50 HALLO = 3500
60 PRINT HALLO
RUN
```

Mit Variablen können auch mathematische Operationen ausgeführt werden. Versuchen Sie das folgende Programm:

```
NEW
10 A = 15
20 B = 95
30 PRINT A + B
```

In Variablen kann das Ergebnis von Berechnungen mit anderen Variablen oder mit Zahlen stehen. Geben Sie das folgende Programm ein und prüfen Sie, ob Sie das Ergebnis abschätzen können, bevor Sie das Programm ausführen:

```
10 NIEDRIG = 5
20 HOCH = 9
30 SUMME = NIEDRIG + HOCH
40 PRINT SUMME
```

Die Summe der Variablen NIEDRIG und HOCH wird in der dritten Variablen, SUMME, angegeben.

Führen Sie das folgende Programm aus, um die verschiedenen Kombinationen von möglichen Zahlen und Variablen zu sehen:

```
10 W = 14.5
20 X = 6.5
30 PRINT (W + X) * 2
40 Y = W - X + 3
50 PRINT Y
60 Z = 3 * Y - 2
70 PRINT Z
```

Benennen von Variablen

Beim Benennen von Variablen gibt es bei Applesoft einige Einschränkungen. Sie werden nachfolgend aufgeführt:

- Ein Variablenname muß mit einem Buchstaben beginnen.
- Bei den Zeichen hinter dem ersten Zeichen kann es sich um eine Mischung aus Buchstaben und Zahlen (keine Symbole und keine Umlaute) handeln.
- Einige Buchstabenkombinationen (die als reservierte Wörter bezeichnet werden) haben für Applesoft eine besondere Bedeutung und dürfen in keinem Variablennamen benutzt werden. (Dies wird im einzelnen in Lektion 3 beschrieben.)
- Ein Name kann maximal 238 Zeichen umfassen. Der Computer erkennt jedoch nur die beiden ersten Zeichen. (Die anderen Zeichen geben nur die Bedeutung der Variablen an.)

Beim Schreiben sehr kurzer und einfacher Programme wird durch Variablen aus einem Buchstaben garantiert, daß ein Variablenname einmalig ist. (Aufgrund der letzten Regel in der Tabelle betrachtet der Computer SOHN und SONNTAG als dieselbe Variable.) Werden jedoch längere Programme geschrieben, so werden Variablennamen empfohlen, mit denen die jeweilige Funktion beschrieben wird.

Wird beispielsweise der Inhalt eines Kreises berechnet, so wird der Wert von pi (π) in dem Programm benötigt. In der Variablen x könnte der Wert von pi (3.141592) stehen. Es ist jedoch sinnvoller, den Variablen bedeutungsvollere Namen zuzuweisen:

```
NEW
10 PI = 3.141592
20 RADIUS = 5
30 INHALT = PI * RADIUS * RADIUS
40 PRINT INHALT
```

Mathematik: $A = \pi R^2$.

Durch die beschreibenden Variablennamen können Sie problemlos erkennen, was das Programm tut, wenn Sie es lesen.

❖ *In numerischen Variablen dürfen nur Zahlen gespeichert werden:* Die Art Variablen, die hier beschrieben werden, werden als **numerische Variablen** bezeichnet. Dies bedeutet, daß sie nur zur Speicherung des Wertes von Zahlen benutzt werden können. In Lektion 3 werden die **Zeichenfolgen-Variablen** beschrieben, in denen beliebige Angaben stehen - Zahlen, Buchstaben, Sonder - zeichen. Erhalten Sie eine Fehlermeldung, wie beispielsweise `TYPE MISMATCH` (Typ stimmt nicht überein), so haben Sie wahrscheinlich versucht, einer numerischen Variablen einen nichtnumerischen Wert zuzuweisen.

Einige Regeln brechen

Am besten versteht man eine Programmierregel, wenn man sie einmal bricht. Brechen Sie jede Regel für die Variablen und sehen Sie, was geschieht. Nur zu! Hier einige Beispiele:

```
NEW
10 PRINT 1V
RUN
10
```

Der Computer dachte, Sie wollten eine *1* drucken und danach den Wert der Variablen *V.* (Alle Variablennamen beginnen mit einem Buchstaben.) Variablen, denen kein Wert zugewiesen wurde, haben automatisch den Wert 0. Eine *1* mit einer *0* daneben entspricht *10.*

```
10 PRINT = 1
RUN
?SYNTAX ERROR IN 10
```

PRINT ist ein reserviertes Wort. Es kann nicht als Variable benutzt werden.

```
10 MIMI = 5
20 MIAMI = 8
30 PRINT MIMI
RUN
8
```

Nur die beiden ersten Zeichen eines Variablennamens sind von Bedeutung. Was Ihren Apple betrifft, so haben Sie `MI` in Zeile 10 den Wert 5 zugewiesen, haben ihn in Zeile 20 jedoch in 8 geändert.

Das Festlegen von bedeutungsvollen und gleichzeitig zulässigen Variablennamen kann am Anfang etwas schwieriger sein. Kommt es also zu einem Programmfehler, so prüfen Sie als *erstes* Ihre Variablennamen.

Zusammenfassung und Überblick

In dieser Lektion wurde beschrieben, wie arithmetische Operationen bei Ihrem Computer ausgeführt und Variablen benutzt werden. Sie haben die Prioritäten kennengelernt und wissen, wie der Computer programmiert werden muß, um erst einmal einfache und danach komplexere arithmetische Operationen auszuführen. Sie wissen, daß Variablen Speicherbereiche sind, in denen Werte stehen, und daß die den Variablen zugewiesenen Namen die Art der Werte wiedergeben sollten. Außerdem haben Sie festgestellt, daß es für das Benennen von Variablen -wie sonst auch- Regeln gibt. Regeln,die am besten erlernt werden, indem sie einmal gebrochen werden.

Lektion 3

Die Außenwelt

Bis jetzt wurden alle Informationen, die in den Computer eingegeben wurden, über Programmzeilen eingegeben. Sollte eine Variable einen bestimmten Wert erhalten, so wurde eine **Zuweisungs-Instruktion** benutzt (wie bei ZAHL = 23). Sie heißt so, da sie der Variablen ZAHL den Wert 23 *zuweist.* Sie, der Programmierer, haben dem Programm den Wert der Variablen gegeben. In dieser Lektion wird beschrieben, wie INPUT benutzt wird, eine Instruktion, mit der das Programm den Wert einer Variablen vom Benutzer erhält. Sie werden erfahren, wie bedeutungsvolle Bedienerhinweise erstellt werden, damit der Benutzer weiß, welche Informationen das Programm benötigt. Außerdem werden die **Zeichenfolgen-Variablen** beschrieben, mit denen Variablen Buchstaben und Sonderzeichen (nicht nur Zahlen) zugewiesen werden.

Darüber hinaus wird der Unterschied zwischen der **sofortigen Ausführung** und der **verzögerten Ausführung** beschrieben. Außerdem werden neue Instruktionen vorgestellt, mit denen der Bildschirm gelöscht (HOME) und eine aktualisierte Auflistung des Programms (LIST) erhalten werden kann.

INPUT

Die INPUT-Instruktion ist das Kernstück der **interaktiven Programmierung** - der Programmierung, bei der Computer und Mensch miteinander sprechen können. Mit INPUT erhält das Programm während der Ausführung Informationen.Das Programm wartet, bis Sie (oder der Benutzer Ihres Programmes) eine Eingabe vornehmen und Return drücken.

Geben Sie das folgende Programm ein und führen Sie es aus. Wird ein Fragezeichen (der INPUT-Bedienerhinweis) auf dem Bildschirm angezeigt, so geben Sie eine Zahl ein und drücken Return:

```
NEW
10 INPUT A
20 PRINT A * 5
```

Ihr Apple Computer druckt die Zahl aus, die Sie hinter dem Fragezeichen eingegeben haben. Haben Sie "3" eingegeben, so würde der Bildschirm folgendermaßen aussehen:

```
?3
15
```

Der Computer zeigt das Fragezeichen automatisch an.

Dies ist genau so, als hätten Sie "A = 3" als eine Programmzeile eingegeben. Sämtliche als Antwort auf den INPUT-Bedienerhinweis eingegebenen Daten werden der **Eingabevariablen** zugewiesen (einer Variablen, deren Wert vom Benutzer zugewiesen wird, im Gegensatz zu der Variablen, deren Wert vom Programmierer zugewiesen wird).

Bedienerhinweise

Das Fragezeichen fordert Sie zu einer Eingabe auf. Aus diesem Tutorium wußten Sie, was Sie eingeben mußten (eine Zahl). Die Benutzer Ihres Programms wissen jedoch wahrscheinlich allein anhand der Bildschirmanzeige nicht, was sie tun müssen. Ein Fragezeichen an sich ist nicht sehr aussagefähig.

Bei Applesoft können beschreibende **Bedienerhinweise** benutzt werden, um dieses Problem zu lösen. Anhand von Bedienerhinweisen weiß der Benutzer eines Computers, was er als nächstes tun muß. Es gibt zwei Möglichkeiten, mit denen angegeben werden kann, was das Programm wünscht. Als erstes kann eine Zeile gedruckt werden, in der die auszuführenden Schritte ausgedruckt werden. Danach wird eine INPUT-Zeile benutzt.

Geben Sie folgendes Programm ein und führen Sie es aus:

```
NEW
10 PRINT "Ich habe schlecht geschlafen. Welches Jahr schreiben wir?"
20 INPUT Jahr
```

Wird das Programm nun ausgeführt, so wissen Sie anhand der Meldung auf dem Bildschirm, daß das Jahr eingegeben werden muß.

Sie können auch mit der INPUT-Instruktion selbst einen Bedienerhinweis ausdrucken. Ein Bedienerhinweis mit INPUT wird fast genau wie ein Bedienerhinweis mit PRINT benutzt. Allerdings wird der Bedienerhinweis auf derselben Zeile wie die INPUT-Instruktion angezeigt:

```
NEW
10 INPUT "Ich habe schlecht geschlafen. Welches Jahr schreiben wir?"; Jahr
                                                  INPUT und Bedienerhinweis.
20 PRINT "Prima. Ich dachte schon, wir haetten "; Jahr + 1
                                                  Hier werden neue Angaben gemacht..
30 PRINT " und ich haette Weihnachten verpasst."
```

(Der Computer muß unbedingt eine Antwort erhalten, wenn er eine Antwort von Ihnen anfordert.) Das Semikolon zwischen dem Anführungszeichen und dem Variablen-namen in Zeile 10 ist wichtig. Ein Semikolon muß angegeben werden, wenn ein Bedienerhinweis mit einer INPUT-Instruktion benutzt wird. Wird ein Semikolon hinter einer INPUT-Instruktion benutzt, so läßt der Apple den Bedienerhinweis in Form eines Fragezeichens weg.

❖ *Einige Tips zur Benutzung von PRINT:* Aus Zeile 20 ergeben sich einige Folgerungen, die Sie selbst ermitteln können. Als erstes beachten Sie:

1. Hinter dem abschließenden Anführungszeichen steht ein Semikolon - anhand des Semikolons weiß BASIC, daß der Wert der Variablen auf derselben Zeile wie das Anführungszeichen angegeben werden muß.

2. Für die Variable Jahr führt Ihr Apple einige kleine Rechenoperationen durch.

Nachfolgend ein Programm mit einigen Beispielen für INPUT-Zeilen mit Bedienerhinweisen:

```
NEW
10 PRINT "FRAGE-SPIELEPROGRAMM"
20 PRINT
30 INPUT "Wie viele Karten hat ein Kartenspiel? "; Karten
40 INPUT "Wie weit ist der Mond von der Erde entfernt? "; ME
50 INPUT "Wie viele Tasten hat die Tastatur? "; Tasten
60 INPUT "Wie viele Tage hat ein Schaltjahr? "; Schaltjahr
```

❖ *Unzulässige Namen und Syntaxfehler:* Das Frage-Programm benutzt beschreibende Variablennamen in sämtlichen Zeilen mit Ausnahme von Zeile 40. Der Variablenname ME ist nicht sehr aussagefähig, jedoch enthält das Wort Mond das reservierte Wort ON (siehe Liste in Anhang B). Enthält das Programm einen Syntaxfehler und wissen Sie nicht warum, so ändern Sie probehalber die Variablennamen.

Weitere Editierarbeiten: Hinzufügen von Zeilen

Gelegentlich müssen Zeilen zu dem Programm hinzugefügt werden. Müssen die neuen Zeilen an das Ende des Programms angefügt werden, so wird einfach eine Zeilennummer eingegeben, die größer ist als die letzte Zeilennummer in dem alten Programm, und mit der Eingabe begonnen. Was aber, wenn eine Zeile in der Mitte des Programms eingefügt werden muß? Kein Problem. Sie brauchen nur eine Zeilennummer einzugeben, die *zwischen den schon vorhandenen Nummern* liegt.

Angenommen, Sie haben folgendes Programm und möchten beispielsweise zwischen den Zeilen 10 und 20 eine Zeile einfügen:

```
NEW
10 PRINT "Nicht vergessen"
20 PRINT "den Hund zu fuettern"
```

Nun möchten Sie noch sagen, wann der Hund gefüttert werden muß. Hierzu braucht nur die folgende Zeile zu dem Programm hinzugefügt zu werden:

```
15 PRINT "puenktlich morgens"
```

Nun führen Sie das Programm aus. Dabei werden Sie feststellen, daß alles in der richtigen Folge angegeben wird.

❖ *Abstände zwischen den Zeilennummern lassen:* Bei allen Beispielprogrammen in diesem Tutorium wurden die Zeilennummern um jeweils 10 erhöht. Hätte das Programm die Zeilennummern 1, 2 anstelle von 10, 20 gehabt, so wäre kein Platz für das Einfügen der neuen Zeile gewesen und Sie hätten das ganze Programm neu eingeben müssen.

Mit HOME aufräumen

Nachdem Sie einige Programme eingegeben und ausgeführt haben, ist der Bildschirm recht voll. Mit der HOME-Instruktion wird der Bildschirm gelöscht und der Cursor in die obere linke Ecke gesetzt (die Ausgangs- oder *Home*-Position des Cursors). Wann immer das Programm auf HOME stößt, löscht es den Bildschirm und setzt den Cursor in die Ausgangsposition.

```
NEW
10 HOME
20 INPUT "WIE VIEL PFUND HAT EIN KILOGRAMM? "; PF
30 HOME
40 INPUT "WIE ALT IST DER BUNDESPRAESIDENT? "; PRAES
RUN
```

Der Bildschirm wird mit jeder neuen Frage gelöscht. Auf diese Weise ist stets klar, was das Programm erwartet, und so gibt es keine Verwirrung zwischen den einzelnen Programmen.

HOME kann auch ohne Zeilennummer benutzt werden, wenn Sie mal eben etwas Ordnung auf dem Bildschirm schaffen wollen. Geben Sie einfach HOME ein und drücken Sie Return.

Versuchen Sie es:

```
HOME
```

Mit HOME wird der Bildschirm gelöscht - der Speicher jedoch *nicht*. Mit HOME werden nur die Zeichen gelöscht, die den Bildschirm füllen. Dieser Befehl hat absolut keine Auswirkung auf den Speicher. (HOME darf nicht mit NEW verwechselt werden.) Nachdem der Bildschirm jedoch mit HOME gelöscht wurde, brauchen Sie eine Möglichkeit, um die Programmzeilen erneut anzuzeigen.

LIST

Geben Sie `LIST` ein und drücken Sie Return, um das Programm erneut anzuzeigen. Versuchen Sie es.

```
LIST
```

Je länger die Programme werden, desto häufiger werden Sie LIST benutzen. Geben Sie das folgende Programm ein, um die verschiedenen Möglichkeiten für die Benutzung von LIST zu testen:

```
NEW
10 HOME
20 PRINT "Und Maria lief und lief "
30 PRINT "den Strand entlang "
40 PRINT "um Peter zu suchen, "
50 PRINT "dem schlecht war, "
60 PRINT "weil er zuviel gegessen hatte."
```

Als erstes führen Sie das Programm aus. Danach listen Sie es auf. Nachdem Sie das Programm aufgelistet haben, versuchen Sie die folgenden Variationen des LIST-Befehls, um zu sehen, was geschieht.

`LIST 40`	Listet nur Zeile 40 auf.
`LIST 40 -`	Listet ab Zeile 40 bis zum Ende des Programms auf.
`LIST - 40`	Listet vom Anfang bis zu Zeile 40 auf.
`LIST 20 - 40`	Listet von Zeile 20 bis 40 auf.

Bei den kleinen Programmen, die Sie bis jetzt geschrieben haben, brauchen Sie all diese Variationen in dem LIST-Befehl nicht. Später jedoch, wenn die Programme so groß werden, daß sie über den Bildschirm hinausreichen, werden Sie wahrscheinlich häufig kleinere Programmsegmente auflisten wollen.

Zeichenfolgen-Variablen

In Lektion 2 wurde beschrieben, wie Variablen mit Zahlen benutzt werden. Variablen können auch mit Text benutzt werden. Variablen, in denen Text steht, werden als **Zeichenfolgen-Variablen** bezeichnet. Die Namen von Zeichenfolgen-Variablen enden immer mit einem Dollarzeichen ($). Sie werden fast genau so definiert (d.h. ihnen werden Werte zugewiesen) wie die numerischen Variablen:

```
NEW
10 HOME
20 Tante$= "Tante Elisabeth"
30 PRINT Tante$
```

Wird dieses Programm ausgeführt, so werden die Wörter *Tante Elisabeth* auf dem Bildschirm angezeigt. Zeile 30 ist gleichbedeutend mit

```
PRINT "Tante Elisabeth"
```

Einer Zeichenfolgen-Variablen kann nahezu jeder Wert zugewiesen werden. Im Gegensatz zu den numerischen Variablen, denen nur Zahlen zugewiesen werden dürfen, können Zeichenfolgen-Variablen Buchstaben, Zahlen, Symbole - sogar Satzzeichen enthalten:

```
NEW
10 HOME
20 UNSINN$= "All diese sinnlosen Zeichen -> %43$,*!:;"
30 PRINT UNSINN$
```

Darauf hat Ihr Computer sämtliche Angaben zwischen den Anführungszeichen in Zeile 20 ausgedruckt. Hier muß jedoch darauf hingewiesen werden, daß Zahlen nicht als Zahlen behandelt werden, wenn sie in Zeichenfolgen-Variablen stehen. Sie werden als Text behandelt - einfach Symbole, eine Folge von Zeichen ohne Bedeutung für den Computer.

Führen Sie das nächste Programm aus, um zu sehen, wie Zahlen als Text behandelt werden:

```
10 HOME
20 A$ = "10"
30 B$ = "20"
40 PRINT A$ + B$
```

Anstatt *30* zu erhalten, haben Sie *1020* erhalten. Das Pluszeichen (+) "addiert" die Zeichenfolgen-Variablen nicht. (Wie könnten auch Buchstaben addiert werden?) Es fügt die Zeichenfolgen einfach zusammen. In der Computerterminologie sagt man: "Sie werden **verkettet**".

Zeichenfolgen-Variablen können ebenfalls mit INPUT benutzt werden. Bedienerhinweise werden mit einer Zeichenfolgen-Variablen INPUT genau wie bei einer numerischen Variablen INPUT benutzt. Bei dem nächsten Programm werden beide Arten von Variablen gemischt:

```
10 HOME
20 INPUT "Wie heissen Sie? "; NAME$
30 INPUT "Geben Sie das Alter ein: "; NUM
40 HOME
50 PRINT NAME$;                    Beachten Sie das Semikolon.
60 PRINT " ist ";                  Vor dem i und hinter dem t steht ein Leerzeichen.
70 PRINT NUM;
80 PRINT "Jahre alt".
```

Um zu sehen, was geschieht, geben Sie einige Buchstaben ein, wenn Ihr Apple nach Zahlen fragt. (Geben Sie beispielsweise `achtzehn` anstelle der Zahl `18` ein.)

Sobald Sie Return drücken, erhalten Sie folgende Fehlermeldung:

```
?REENTER
```

Dies bedeutet einfach, daß das Programm eine Zahl erwartet und etwas anderes erhalten hat. Tun Sie, worum Sie gebeten werden - geben Sie eine Zahl ein (der Computer ist immer so pingelig!), und alles läuft einwandfrei.

Wiederholung der Regeln für Variablen

Falls Sie sie vergessen haben sollten, nachfolgend noch einmal die Regeln für das Benennen von Variablen. Die letzte Regel gilt nur für Zeichenfolgen-Variablen:

- Ein Variablenname muß mit einem Buchstaben beginnen.
- Auf das erste Zeichen können Buchstaben oder Zahlen folgen.
- Ein Name kann maximal 238 Zeichen umfassen, der Computer erkennt jedoch nur die beiden ersten Zeichen. (Mit den anderen Zeichen wird angegeben, wofür die Variable benutzt wird.)
- Bestimmte Buchstabenkombinationen (sogenannte reservierte Wörter) können in keinem Teil eines Variablennamens benutzt werden. Die reservierten Wörter werden in Anhang B aufgeführt.
- Sämtliche Namen von Zeichenfolgen-Variablen enden mit "$".

Austesten

Das Gesetz der Serie gilt bei der Programmierung umso mehr. (Dasselbe gilt für den Satz "Ein Fehler kommt selten allein"; dies wird jedoch in einem späteren Tutorium behandelt.) Sowohl erfahrene Programmierer als auch Anfänger machen bei der Programmierung viele kleine Fehler. Das Austesten eines Programms (d.h. das Verfolgen und Beseitigen von Fehlern) ist ein normaler Bestandteil beim Schreiben eines Computerprogramms. Nur zu oft ist es der Hauptteil. Deshalb gibt Ihr Computer Fehlermeldungen.

Beim Austesten von Programmen ist der Unterschied zwischen **sofortiger** und **verzögerter Ausführung** von besonderer Bedeutung. Wird `RUN`, `NEW` oder `LIST` ohne eine Zeilennummer eingegeben, so führt der Computer den gewünschten Schritt sofort nach Drücken von Return aus. Dies wird als sofortige Ausführung bezeichnet. Wird ein Programm mit Zeilennummern geschrieben, so verzögert der Computer die Ausführung, bis der entsprechende Befehl gegeben wird. Dies wird als verzögerte Ausführung bezeichnet. Die sofortige Ausführung ist beim Austesten von Programmen von besonderem Nutzen.

Geben Sie beispielsweise das folgende Programm ein und führen Sie es aus:

```
NEW
10 HOME
20 MONETEN$ = "DM 1.000"
30 PRINT MONETEN$
```

Hier erhalten Sie `?SYNTAX ERROR IN 20` anstelle der erwarteten DM 1.000. Listen Sie Zeile 20 auf und Sie werden eine Überraschung erleben:

```
20 M ON ETEN$ = "DM 1.000"
```

Was ist mit `M ON ETEN$` geschehen? Es wurde aufgeteilt. Geben Sie

```
MONETEN$ = "DM 1.000"
```

ein. Sobald Sie Return drücken, erhalten Sie einen Syntaxfehler. Sie haben ein reserviertes Wort (ON) in dem Variablennamen benutzt. Aus der Programmauflistung sehen Sie, daß ON in den Zeilen 20 und 30 von `M ON ETEN$` getrennt wurde. Sie können das Programm mit einem anderen Variablennamen neu schreiben. Als erstes testen Sie jedoch den neuen Namen mit der sofortigen Ausführung.

Versuchen Sie folgendes:

```
GELD$ = "DM 1.000"
```

Diesmal kam keine Fehlermeldung. Dies bedeutet, daß `GELD$` als Variablenname zulässig ist. In diesem Fall dauert die Änderung des Programms nur einige wenige Sekunden; Sie haben `MONETEN$` nur einmal benutzt. Denken Sie jedoch nun an ein wesentlich größeres Programm, in dem Sie `MONETEN$` 25 oder 30 mal benutzt haben - es würde recht lange dauern, jedes Auftreten von `MONETEN$` in `GELD$` zu ändern. Es geht wesentlich schneller, mögliche Fehler mit der sofortigen Ausführung zu testen, als das Programm bei jedem Auftreten eines Fehlers neu zu schreiben.

Das Geheimnis des erfolgreichen Austestens ist das Eingrenzen des Problems. Bei einigen Fehlermeldungen wird die Zeilennummer angegeben, in der der Computer den Fehler entdeckt hat. Dadurch kann das Problem schnell eingegrenzt werden. Testen Sie das mögliche Problem im Sofort-Modus, wie in dem Beispiel mit `MONETEN$` und `GELD$` gezeigt. Korrigieren Sie den Fehler in dem Programm und führen Sie das Programm erneut aus, um festzustellen, ob weitere Fehler auftreten. Kommt es zu keinen weiteren Fehlern, so wurde das Austesten erfolgreich vorgenommen - zumindest was die Variablennamen betrifft.

Im weiteren Verlauf dieses Tutoriums werden Sie noch häufig die sofortige Ausführung benutzen. Experimente sind der Schlüssel zum Erfolg. Versuchen Sie alles zuerst mit der sofortigen Ausführung; so erleben Sie keine unangenehmen Überraschungen.

Zusammenfassung und Überblick

In dieser Lektion haben Sie gelernt, daß Sie mit der INPUT-Instruktion Informationen vom Benutzer erhalten können, während die Programme ausgeführt werden. Mit INPUT müssen unbedingt aussagefähige Bedienerhinweise benutzt werden. Auf diese Weise wissen die Benutzer Ihrer Programme, was sie eingeben müssen. Aussagefähige Bedienerhinweise sind für den Benutzer der Programme das, was aussagefähige Variablennamen für Sie, den Programmierer, sind.

Außerdem wurden die Zeichenfolgen-Variablen vorgestellt. Sie haben gesehen, daß sie wie die numerischen Variablen benutzt werden und aussehen. Allerdings enden die Zeichenfolgen-Variablen mit einem $, und ihre Werte stehen in einer Programmzeile in Anführungszeichen.

Mit der HOME-Instruktion wird der Bildschirm gelöscht. Mit LIST können sämtliche Zeilen des Programms im Speicher aufgelistet werden, um das Austesten des Programms zu vereinfachen.

Außerdem wurde beschrieben, daß viele Programmierbefehle mit der sofortigen Ausführung benutzt werden können, um das Austesten der Programme zu vereinfachen.

Lektion 4

Benutzung von Disketten und Druckern

Sobald Sie längere und bessere Programme schreiben, werden Sie sie zur späteren Benutzung sichern wollen. In dieser Lektion wird beschrieben, wie Programme auf Disketten gespeichert und wieder in den Speicher zurückgeholt werden.

Drei verschiedene Arten von Speichern (**RAM**, **ROM**, **Disketten**) werden vorgestellt, wobei besonderer Nachdruck auf den Diskettenspeicher gelegt wird. Es wird beschrieben, wie ein Programm mit SAVE auf einer Diskette gespeichert und mit LOAD zurück in den Hauptspeicher geladen wird. Außerdem wird eine Liste der Programme auf einer Diskette mit CAT erstellt. Ferner wird beschrieben, wie veraltete Programme auf einer Diskette mit DELETE gelöscht werden.

Schließlich wird beschrieben, wie mit PR#1 eine Version des Programms auf Papier anstatt auf den Bildschirm ausgegeben wird, und wie mit PR#0 der Bildschirm wieder benutzt wird. Die Lektion wird wieder mit einem Überblick über die behandelten Themen abgeschlossen.

Speicher des Computers

RAM steht für ***R****andom-****A****ccess* ***M****emory* (Schreib-/Lese-Speicher). Damit wird der **Hauptspeicher** eines Computers bezeichnet. Wird der Computer das erste Mal eingeschaltet, so enthält dieser Speicher nichts Sinnvolles. Schreiben Sie ein Programm oder weisen Sie den Computer an, ein auf einer Diskette gespeichertes Programm zu laden, so werden diese Informationen in den RAM gebracht. Wird der Computer ausgeschaltet, so sind sämtliche Informationen im RAM verloren.

ROM ist die Abkürzung für ***R****ead-****O****nly* ***M****emory* (Nur-Lese-Speicher). Hier handelt es sich um einen Speicher, in dem die Informationen permanent gespeichert sind. Die Applesoft BASIC-Sprache ist in diesem Speicher gespeichert. Wird der Computer ausgeschaltet, so bleibt die Sprache im ROM (dies gilt allerdings nicht für Ihr Programm). In diesem Speicher wird keine Benutzereingabe gespeichert.

Auf einer **Diskette** werden Programme gesichert. Diskettenlaufwerke (die Geräte, in die die Disketten eingelegt werden) können in einem gewissen Sinn mit Magnetbandgeräten verglichen werden. Bei einem Magnetbandgerät wird in das Mikrofon gesprochen und die Stimme auf dem Magnetband aufgezeichnet. Danach wird das Band zurückgespult und die Stimme abgehört. Der Computer arbeitet ähnlich. Allerdings werden keine Bandgeräte benutzt, um den Inhalt des RAM auf Band zu sichern, sondern Diskettenlaufwerke, um Informationen auf Disketten zu sichern. Sobald ein Programm auf Diskette steht, kann es immer wieder "abgespielt" werden.

Sie brauchen sich keine Gedanken um die technischen Einzelheiten der RAM, ROM und Disketten zu machen. Allerdings ersparen Sie sich viel Mühe, wenn Sie daran denken, daß beim Ausschalten Ihres Computers sämtliche Daten im RAM verloren gehen.

Dateien und Inhaltsverzeichnisse

Die meisten gut organisierten Menschen legen Dokumente in Akten ab, um sie wiederzufinden. Dasselbe gilt für die Aufzeichnungen des Computers. Auf einer Diskette gespeicherte Programme werden als **Dateien** bezeichnet. Es gibt verschiedene Arten von Dateien. Sie brauchen sich hier jedoch nur mit den Programmdateien zu beschäftigen - so werden auf Disketten gesicherte Programme bezeichnet.

Werden die in einem Aktenschrank abgelegten Akten in einer Liste oder einem **Inhaltsverzeichnis** aufgezeichnet, so können sie bei Bedarf schneller gefunden werden. Im wesentlichen tut Ihr Computer dasselbe, wenn Sie ein Programm auf einer Diskette sichern. Sie speichern das Programm mit dem SAVE-Befehl, und der Name des Programms wird in ein Inhaltsverzeichnis gesetzt. Soll ein Programm benutzt werden, so wird es mit dem CAT-Befehl in dem Inhaltsverzeichnis der Diskette gesucht, um sicherzustellen, daß es auch wirklich dort vorhanden ist. Danach wird es mit dem LOAD-Befehl rückübertragen.

❖ *Befehle im Gegensatz zu Instruktionen - eine Frage der Terminologie:* Im letzten Absatz wurde der Ausdruck *Befehl* verschiedentlich benutzt. Ein Befehl kann insofern mit einer Instruktion verglichen werden, als er den Computer anweist, eine bestimmte Funktion auszuführen. Der Unterschied zwischen einem Befehl und einer Instruktion liegt nahezu ausschließlich darin, wann der Computer die gewünschte Aktion ausführt. Im wesentlichen ist ein Befehl eine Anweisung, die der Computer sofort ausführt. Eine Instruktion ist eine Anweisung, deren Ausführung verzögert wird. Dies ist nur eine Frage der Terminologie.

Sichern von Programmen

Die Speicherung eines Programms auf einer Diskette ist die einfachste Sache der Welt. Sie geben den SAVE-Befehl aus und weisen dem Programm einen Namen zu, unter dem Sie es später wieder von der Diskette aufrufen können.

Um etwas Übung zu bekommen, geben Sie als erstes dieses Programm ein:

```
NEW
10 PRINT "Dies ist mein erstes gesichertes Programm."
20 PRINT "Ich bin sehr stolz darauf,"
30 PRINT "(oder werde es sein, wenn ich es wieder aufrufen kann)."
```

Nun brauchen Sie noch einen Namen. Nachfolgend einige Regeln für das Benennen eines Programms.

- Der Name eines Programms kann maximal 15 Zeichen umfassen.
- Der Name muß mit einem Buchstaben beginnen.
- Der Name darf keine Umlaute oder "ß"enthalten.
- Buchstaben, Zahlen und Punkte können in Dateinamen benutzt werden. Allerdings können keine anderen Zeichen und insbesondere keine Leerzeichen benutzt werden. Es können sowohl Groß- als auch Kleinbuchstaben benutzt werden, der Computer setzt jedoch alle Buchstaben in Großbuchstaben um.
- Sämtliche Dateinamen auf einer bestimmten Diskette müssen einmalig sein. Hier gelten jedoch alle Zeichen in dem Namen und nicht nur die beiden ersten. Um reservierte Wörter brauchen Sie sich keine Gedanken zu machen. Deshalb sollte die Zuweisung verschiedener Dateinamen kein Problem darstellen.
- Der Name sollte die Funktion des Programms angeben.

Nachfolgend einige zulässige Dateinamen:

SCHECKBUCH

ADDIER.PROGRAMM

AH.1UNDAH.2

TEST.23.10.

Folgende Namen hingegen sind *unzulässig*:

Unzulässiger Name	Grund
1EINS	Beginnt mit einer Zahl.
DIES.PROGRAMM!	Ausrufungszeichen sind unzulässig.
.PUNKT	Beginnt mit einem Punkt.
EIN.SUPERTOLLES.PROGRAMM	Viel zu lang.
ERSTE KLASSE	Ein Leerzeichen.
ÜBUNG	Enthält einen Umlaut.

(Viele Benutzer verwenden in Dateinamen an der Stelle, an der sie Leerzeichen benutzen würden, Punkte.)

Nun sichern Sie Ihr Programm auf einer Diskette. Sie können jeden beliebigen gültigen Namen benutzen; ERSTE.DATEI scheint ein geeigneter Name zu sein.

Geben Sie folgende Zeile ein und drücken Sie Return:

```
SAVE ERSTE.DATEI
```

Das Laufwerk arbeitet. Sobald die Diskette stoppt, ist eine Kopie des Programms sicher auf der Diskette gespeichert. Beachten Sie das Wort - Kopie. Die Speicherung eines Programms auf Diskette hat keine Auswirkungen auf den Speicherinhalt des Computers.

Geben Sie `LIST` ein und drücken Sie Return. Sie werden feststellen, daß das Programm noch immer da ist.

Lesen des Inhaltsverzeichnisses und Rückübertragen eines Programms

Nachdem Sie das Programm auf die Diskette gesichert haben, geben Sie `NEW` ein und betätigen Return. Nun wissen Sie sicher, daß nichts mehr im Speicher steht. (Geben Sie `LIST` ein und drücken Sie Return zur Bestätigung.)

Mit dem CAT-Befehl können Sie die Dateien auf der Diskette betrachten. Sie erhalten eine Liste sämtlicher Dateien auf der Diskette.

Geben Sie folgenden Befehl ein und drücken Sie Return:

```
CAT
```

Angenommen, auf der Diskette sind keine weiteren Programme mehr vorhanden, so sieht der Bildschirm folgendermaßen aus:

```
]CAT

/TEST

NAME              TYPE      BLOCKS     MODIFIED

ERSTE.DATEI        BAS           1     <NO DATE>

BLOCKS FREE:      272       BLOCKS USED:      8

]
```

(Selbstverständlich sieht der Bildschirm anders aus, wenn noch andere Programme auf der Diskette stehen.) Nun steht das Programm ERSTE.DATEI im Inhaltsverzeichnis. (Im Benutzerhandbuch für den Computer wird die Bedeutung der restlichen Anzeigen beschrieben.) Als nächstes wird das Programm rückübertragen. Hierzu wird ein neuer Befehl benötigt, LOAD.

Dieser Befehl wird eingegeben und Return gedrückt:

```
LOAD ERSTE.DATEI
```

Nun hören Sie wieder das Laufwerksgeräusch. Danach werden der Bedienerhinweis und der Cursor wieder angezeigt. Dies bedeutet, daß das Programm erfolgreich in den Speicher geladen wurde.

Um zu prüfen, ob es sich um das gesicherte Programm handelt, listen Sie es auf:

```
LIST
```

Das Programm wird genau in der Form angezeigt, in der es gesichert wurde.

❖ *LOAD gleich NEW:* Wird ein Programm geladen, so löscht der Computer zuerst ein eventuell schon im Speicher stehendes Programm. Dies bedeutet, daß keine zwei Programme miteinander gemischt werden können. (Zwar können zwei Programme kombiniert werden, diese Technik ist für dieses Tutorium jedoch etwas zu hoch entwickelt.) Stellen Sie sich einfach vor, daß vor LOAD automatisch NEW gemacht wird.

Aufräumen

Wird beim Schreiben von Programmen mit Bedacht vorgegangen, so werden Sie während des Schreibens verschiedene Versionen sichern. So haben Sie unter Umständen folgende Programme auf der Diskette gesichert:

FOTOS.V1

FOTOS.V2

FOTOS.V3

Sind Sie sicher, daß die letzte Version des Programms, FOTOS.V3, als einzige benutzt werden soll, so können Sie die anderen Versionen löschen und Platz auf der Diskette schaffen. Dateien werden mit dem DELETE-Befehl gelöscht.

Um FOTOS.V1 zu löschen, geben Sie folgendes ein:

`DELETE FOTOS.V1` Return drücken.

Wieder hören Sie das Laufwerksgeräusch, und FOTOS.V1 gehört nur noch der Erinnerung (des Menschen, nicht des Computers) an. Stellen Sie sich DELETE einfach als Gegenteil von SAVE vor und benutzen Sie dasselbe Format.

❖ *Delete kann nicht rückgängig gemacht werden:* DELETE gilt für immer. Nachdem ein Programm von der Diskette gelöscht wurde, ist es weg. Bevor Sie DELETE benutzen, sollten Sie sicher sein, daß das Programm auch wirklich gelöscht werden soll.

Für Benutzer von Druckern: Ausdrucken der Auflistungen

Bis jetzt wurde das Programm immer an den Bildschirm und die Diskette gesendet. Sie können das Programm (und alle anderen Eingaben) auch an den Drucker senden. Das Ausdrucken eines Programms, insbesondere eines langen Programms, ist beim Austesten äußerst hilfreich. Mit wachsender Erfahrung werden Sie feststellen, wie wahr dies ist.

Um ein Programm auf dem Drucker aufzulisten, gehen Sie folgendermaßen vor:

1. Prüfen Sie, ob der Drucker richtig an den Computer angeschlossen ist.
2. Prüfen Sie, ob das Papier richtig eingelegt ist.
3. Vergewissern Sie sich, ob der Drucker eingeschaltet ist.
4. Geben Sie PR#1 ein und drücken Sie Return.

(Wird eine der drei ersten Instruktionen nicht befolgt, so scheint der Computer stehengeblieben zu sein.) Durch den Befehl `PR#1` werden sämtliche Angaben, die ansonsten auf dem Bildschirm angezeigt werden, an den Drucker geleitet. Geben Sie nach Eingabe eines `PR#1`-Befehls `LIST` ein, so druckt der Drucker die Auflistung aus - es sei denn, Sie haben `LIST` falsch geschrieben - in diesem Fall wird die Meldung über den Syntaxfehler ausgedruckt).

Soll die Ausgabe des Computers wieder auf dem Bildschirm angezeigt und der Drucker nicht weiter benutzt werden, so geben Sie folgenden Befehl ein:

```
PR#0
```

und drücken Return. Der Befehl wird auf der gedruckten Seite ausgegeben. Nachfolgende Befehle und Auflistungen werden jedoch wieder auf dem Bildschirm angezeigt.

In längeren Programmen sind Fehler oft schwierig zu finden, insbesondere, wenn die Auflistung so lang ist, daß sie vom Bildschirm rollt. Durch das Ausdrucken der Auflistungen kann viel Zeit beim Austesten gespart werden.

Geben Sie folgendes Programm ein und versuchen Sie, es auf dem Drucker aufzulisten:

```
NEW
10 HOME
20 PRINT "Dieses Programm wird auf dem Drucker aufgelistet."
30 PRINT "Hat es einen Fehler, so wird"
40 PRINT "der Drucker helfen, ihn zu finden."
PR#1
LIST
```

Der Drucker listet den Programmtext auf.

Bevor Sie den Drucker mit `PR#0` ausschalten, führen Sie das Programm aus, um zu sehen, was geschieht. Danach geben Sie `PR#0` ein, um wieder den BASIC-Bedienerhinweis ( ]) auf dem Bildschirm zu erhalten.

Praktische Übungen

In dieser Lektion brauchten Sie weniger zu lernen als in den drei vorherigen Lektionen. Benutzen Sie die restliche Zeit, um einige Programme zu schreiben, in denen alle bislang vorgestellten Instruktionen und Operatoren benutzt werden. Hier eine kurze Gedächtnisstütze:

Instruktionen

HOME	INPUT	PRINT

Operatoren

+	-	*
/	(	)

Befehle

CAT	DELETE	LIST
LOAD	NEW	PR#0
PR#1	RUN	SAVE

Grundlagen

Aussagefähige Namen	Bedienerhinweise
Numerische Variablen	Priorität
Sofortige und verzögerte Ausführung	
Zeichenfolgen-Variablen	
Zeilennummern in Zehnerschritten	

Zusammenfassung und Überblick

In dieser Lektion haben Sie gelernt, wie Programme mit dem SAVE-Befehl auf Diskette gesichert und mit LOAD zurückgeholt werden. Außerdem wurde beschrieben, wie Programmen Namen zugewiesen werden und welche Zeichen in einem Namen zulässig und welche nicht zulässig sind. Sie haben gesehen, daß Sie mit CAT eine Auflistung sämtlicher Dateien auf der Diskette erhalten. Mit PR#1 werden sämtliche Daten, die normalerweise auf dem Bildschirm angezeigt werden, auf dem Drucker ausgegeben. (Durch PR#0 werden die Informationen wieder an den Bildschirm gesendet.)

Lektion 5

Schleifen und Bedingungen

In den ersten Lektionen haben Sie die Grundlagen der BASIC-Programmierung erlernt. Nun wollen wir etwas weiter gehen. In dieser Lektion werden drei sehr wichtige Funktionen beschrieben: Schleifen, Vergleichsoperatoren und Bedingungen. Außerdem werden einige BASIC-Abkürzungen angeführt, mit denen die Programmierung einfacher wird. Darüber hinaus werden noch weitere nützliche Instruktionen vorgestellt.

Schleifen

Mit einer Schleife wird derselbe Teil eines Programms mehr als ein Mal ausgeführt. Angenommen, Sie möchten zehn Namen mit INPUT holen und nacheinander auf dem Bildschirm ausdrucken. Es wäre wesentlich einfacher, den Teil des Programms mit der INPUT-Instruktion zu **wiederholen**, als zehn separate Zeilen mit INPUT zu schreiben:

```
NEW
10 HOME
20 INPUT "Ich brauche einen Namen: "; NAME$
30 PRINT NAME$
40 ...
```

Wie kommen Sie nun wieder in Zeile 20?

Sie brauchen eine Instruktion, mit der das Programm wieder zu Zeile 20 zurückgeht, um einen weiteren Namen zu holen. Diese Instruktion ist GOTO.

GOTO

Mit der GOTO-Instruktion wird das Programm angewiesen, zu der angegebenen Zeile zu gehen. Mit diesem Programm wird der Bildschirm gelöscht und danach zu Zeile 40 gegangen (oder **verzweigt**), anstatt mit Zeile 30 weiterzuarbeiten:

```
NEW
10 HOME
20 GOTO 40
30 PRINT "Hallo! Ich dachte, ich waere die naechste!"
40 PRINT "Ich werde als einzige Zeile angezeigt!"
```

Dies wird nie gedruckt!

Hier ein weiteres Beispiel. Geben Sie das erste Programm dieser Lektion ein, diesmal jedoch mit

```
40 GOTO 20
```

in der letzten Zeile. Danach listen Sie es auf. Es sollte folgendermaßen aussehen:

```
10 HOME
20 INPUT "Ich brauche einen Namen: "; NAME$
30 PRINT NAME$
40 GOTO 20
```

Dieses Programm fragt wiederholt nach einem Namen und druckt dann den eingegebenen Namen aus. Das Programm wiederholt dies endlos, solange jemand Namen eingibt (oder bis jemand den Stecker herauszieht).

Sobald das Programm zu Zeile 40 kommt, geht es zurück zu Zeile 20.

❖ *Endlosschleifen:* Hier haben wir eine Endlosschleife. Gelegentlich sind Endlosschleifen recht hilfreich - dies trifft hier jedoch nicht zu. Um die Schleife zu verlassen, bevor keine Namen mehr vorhanden sind (oder Sie die Geduld verlieren), drücken Sie die Tasten Control und C gleichzeitig. Danach lassen Sie sie sofort wieder los und drücken Return. Dies wird als Betätigung von Control-C bezeichnet. Diesem Ausdruck werden Sie häufig begegnen, wenn Sie Computerbücher und Magazine lesen. Nach Drücken von Control-C gibt der Computer:

```
BREAK IN 20
```

an. Diese Meldung bedeutet, daß Sie in Zeile 20 des Programms "eingebrochen" sind. Hängt sich ein Programm auf, so kann die Kontrolle in vielen Fällen nur wiedererlangt werden, indem Control-C gedrückt wird.

Mit diesem Programm wird das Problem der Eingabe vieler Namen gelöst, ohne immer wieder INPUT-Zeilen schreiben zu müssen. Allerdings ist es nicht mehr unter Kontrolle. Sie benötigen eine Möglichkeit (eine andere Möglichkeit als Control-C), die Schleife des Programms zu unterbrechen, sobald Sie genügend Namen haben.

Bedingte Verzweigung mit IF...THEN

BASIC verfügt über eine zweiteilige Instruktion namens IF...THEN. Mit dieser Instruktion kann das Programm Entscheidungen fällen - und dies ist genau das, was Sie brauchen, um das Problem der Endlosschleifen zu lösen. IF...THEN hat das folgende allgemeine Format:

IF *<etwas wahr ist>* THEN *<eine bestimmte Aktion ausführen>*

Mit einer IF...THEN Instruktion wird entschieden, ob eine Bedingung wahr ist oder nicht. Ist das, was im ersten Teil zwischen den Wörtern *IF* und *THEN* (der sogenannten **Bedingung**) steht, wahr, so führt der Computer die Angabe hinter *THEN* aus. Ist die Bedingung *nicht* wahr, so ignoriert das Programm sämtliche Angaben hinter *THEN* und geht zur nächsten Zeile.

Um zu sehen, wie sich dies in der Praxis auswirkt, fügen Sie zwei Zeilen zu dem Programm mit der Endlosschleife hinzu:

```
25 IF NAME$ = "genug" THEN GOTO 50
50 PRINT "Und damit ist die Namensliste beendet."
```

Nachfolgend die ganze Auflistung:

```
10 HOME
20 INPUT "Ich brauche einen Namen: "; NAME$
25 IF NAME$ = "genug" THEN GOTO 50
30 PRINT NAME$
40 GOTO 20
50 PRINT "Und damit ist die Namensliste beendet."
```

Führen Sie das Programm nun aus. Nachdem Sie einige Namen eingegeben haben, geben Sie "genug" ein und das Programm wird beendet.

Ausbauen des Modells

In dem Modell IF *<etwas wahr ist>* THEN *<eine bestimmte Aktion ausführen>* in dem vorherigen Beispiel ist die Bedingung *NA$ = "genug"* . Erhielte NA$ irgendwelche Werte *außer* "genug", so würde die Schleife des Programms fortgesetzt. Sobald der Name "genug" eingegeben worden ist, verzweigt sich das Programm zur letzten Zeile. Die Verzweigung entspricht dann dem Teil "eine bestimmte Aktion ausführen".

❖ *Einen Sicherungsplan aufstellen:* Während die Programme eingegeben werden, sollten Sie sich angewöhnen, sie vor der Ausführung auf einer Diskette zu sichern. Werden sie dann weiterentwickelt und geändert, so müssen sie häufig gesichert werden - etwa alle zehn Minuten. Es gibt Situationen, bei denen Sie das Problem nicht mit Control-C lösen können (beispielsweise wenn jemand versehentlich den Netzschalter ausschaltet). Werden die Programme häufig gesichert, so brauchen die letzten Änderungen nicht neu erstellt und neu eingegeben zu werden.

Vergleichsoperatoren

Nachfolgend einige Beispiele für IF...THEN Instruktionen. Beachten Sie die Bedingungen sorgfältig. Sie werden auf einige bisher unbekannte Symbole stoßen:

```
IF NA$ = "ENDE" THEN GOTO 100
IF A$ <> "APPLE" THEN PRINT "VERLOREN!"
IF SUM > 10 THEN X = 5
IF COUNT < 100 THEN GOTO 20
```

<> bedeutet "ungleich".
> bedeutet "größer als".
< bedeutet "kleiner als".

Diese kleinen spitzen Klammern werden als **Vergleichsoperatoren** bezeichnet. Mit ihnen wird ein *Vergleich* beschrieben, der zwischen zwei Dingen vorgenommen wird. Nachfolgend eine Tabelle, in der alle Vergleichsoperatoren und ihre Bedeutung angegeben werden:

Operator	Bedeutung
>	Größer als
<	Kleiner als
=	Gleich
<>	Ungleich
>=	Nicht kleiner als
<=	Nicht größer als

Die nächsten beiden Programme enthalten einige Beispiele für Vergleichsoperatoren, sowie GOTO und IF...THEN Instruktionen. Sie enthalten einige Herausforderungen, stellen eine oder zwei neue Instruktionen vor und weisen auf einige BASIC-Abkürzungen hin.

Vergleichen von Werten: Dieses Programm fordert vom Benutzer zwei Zahlen an und sagt Ihnen dann, welche Zahl die niedrigere ist. Das Programm enthält einige Überraschungen, damit es Ihnen nicht langweilig wird.

Als erstes geben Sie das Programm ein. Danach versuchen Sie *vor* der Ausführung festzustellen, was geschehen wird. Schließlich führen Sie das Programm aus und sehen, ob Sie Recht hatten.

```
NEW
10 HOME
15 PRINT "Um das Programm zu beenden, geben Sie eine 0 als erste Zahl ein."
20 INPUT "Geben Sie die erste Zahl ein:  "; N1
25 IF N1 = 0 THEN END
30 INPUT "Geben Sie die zweite Zahl ein:  "; N2
35 IF N1 > N2 THEN GOTO 100
40 IF N1 < N2 THEN GOTO 200
45 PRINT "Diese Zahlen sind gleich!"
50 GOTO 20
100 PRINT N2;" ist kleiner als "; N1
110 GOTO 20
200 PRINT N1;" ist kleiner als "; N2
210 GOTO 20
```

Wie kommt dies?

Hier einige Fragen, die vor dem Weiterlesen berücksichtigt werden müssen:

1. In Zeile 25 gibt es eine neue Instruktion - END. Welche Funktion hat sie?
2. In Zeile 45 wird die Meldung nur ausgedruckt, wenn beide in den Zeilen 20 und 30 eingegebenen Zahlen gleich sind. Warum?

Funktionsweise des Programms: Aus Zeile 15 geht hervor, wie Sie das Programm ohne Control-C stoppen können. Mit der END-Instruktion in Zeile 25 wird das Programm gestoppt - jedoch nur, wenn Sie eine 0 eingeben. Zeile 45 wird nur ausgeführt, wenn die Werte für N1 und N2 identisch sind. Die Erklärung hierfür geht aus den beiden vorhergehenden Zeilen hervor. Zeile 35 verzweigt zu einem Teil des Programms, wenn der Wert von N2 kleiner ist als N1. Zeile 40 geht zu einem anderen Teil des Programms, wenn das Gegenteil der Fall ist. Logisch wie er ist, fährt Ihr Computer nur mit der nächsten Zeile (Zeile 45) fort, wenn es keinen Grund dafür gibt, dies nicht zu tun - in diesem Fall, wenn beide Werte identisch sind.

Zuweisung von Variablen: Aus dem nächsten Programm geht hervor, wie IF...THEN Variablen verschiedene Werte zuweisen kann. In diesem Fall entsprechen die Werte verschiedenen Wörtern. (Es könnten auch Zahlen sein.)

Geben Sie das Programm ein. Bevor Sie es ausführen, beantworten Sie jedoch folgende Fragen:

1. Warum all diese Anführungszeichen?
2. Was ist seltsam an Zeile 80?
3. Wofür wird Zeile 80 überhaupt benutzt?

Ermitteln Sie als erstes die Herausforderungen dieses Programms, bevor Sie dieses zoologisch fragliche Programm ausführen:

```
NEW
10 HOME
20 ? "1. SCHWIMMT"
30 ? "2. LAEUFT"
40 ? "3. FLIEGT"
50 PRINT
60 PRINT "Stellen Sie sich ein Tier vor. Danach waehlen"
65 ? "Sie eine Zahl, mit der am besten beschrieben wird, "
70 INPUT "wie sich Ihr Tier bewegt. "; ZAHL
80 IF ZAHL > 3 THEN 10
90 IF ZAHL < 1 THEN 10
100 IF ZAHL = 1 THEN TIER$ = "Fisch"
110 IF ZAHL = 2 THEN TIER$ = "Saeugetier"
120 IF ZAHL = 3 THEN TIER$ = "Vogel"
130 PRINT
200 PRINT "Ich wette, Ihr Tier ist ein "; TIER$
```

Diese Fragezeichen sind eine abgekürzte Eingabe für PRINT. Werden bei jeder Benutzung von PRINT vier Tastenanschläge gespart, so sparen Sie wesentlich mehr Zeit, als Sie denken. Bei der Auflistung des Programms wird jedes Fragezeichen in PRINT umgewandelt.

Zeile 80 ist insofern seltsam, als das Wort GOTO ausgelassen wird. Es stellt sich heraus, daß jede der folgenden Formen für die GOTO-Instruktion innerhalb einer IF...THEN Instruktion benutzt werden kann:

IF ZAHL > 3 THEN GOTO 10

IF ZAHL > 3 THEN 10

IF ZAHL > 3 GOTO 10

Anders ausgedrückt, Sie können THEN oder GOTO weglassen - jedoch nicht beide.

Mit den Zeilen 80 und 90 soll verhindert werden, daß ein Benutzer des Programms eine Zahl eingibt, die außerhalb des ausgewählten Bereichs liegt. Solche Fangstellen geben den Benutzern eine weitere Chance, falls sie einen Fehler begehen (was nur allzu menschlich ist).

Benutzung von REM für Bemerkungen

Mit der REM-Instruktion können Sie Hinweise für sich selbst über die Funktion eines Programms schreiben. Mit ihr können diese Hinweise in dem Programm aufgenommen werden. Die Hinweise werden nur beim Auflisten des Programms angegeben. Benutzer können sie nicht sehen, wenn sie das Programm ausführen.

So haben Sie beispielsweise mit REM-Instruktionen stets Informationen über das Programm. Sie geben Auskunft, was das Programmsegment tut:

```
100 REM    **************************************
110 REM    Das große deutsche Computerprogramm
115 REM    von Bretbeck-Overschneider
120 REM    Version 16.5
125 REM    25. Juni 1986
130 REM    ***************************************
135 REM    Bildschirm löschen
140 HOME
145 KOMMENTAR$ = "REM-Kommentare werden nicht auf dem Bildschirm angezeigt."
150 REM    Eine Meldung auf dem Bildschirm ausdrucken
155 PRINT KOMMENTAR$
160 ....
```

REM-Instruktionen sind Hinweise für Benutzer und nicht für die Computer. REM-Instruktionen haben keine Auswirkung auf das Programm. Sobald das Programm zu einer REM-Instruktion kommt, ignoriert es die Instruktion (und sämtliche Angaben hinter der Instruktion auf derselben Zeile) und geht direkt zur nächsten Zeile.

❖ *Programmnamen in eine REM-Zeile setzen:* Setzen Sie an den Anfang des Programms eine REM-Zeile, in der der Name des Programms angegeben wird. Wird danach das Programm geändert und soll die neue Version auf einer Diskette gesichert werden, so wissen Sie immer, welchen Namen Sie benutzen müssen.

Zeit zum Üben

In dieser Lektion haben Sie sehr viel gelernt. Bevor Sie fortfahren, sollten Sie etwas mit diesen Dingen experimentieren. Gehen Sie zurück und ändern Sie die Beispielprogramme. Versuchen Sie einige Programme "aufzuknacken". Finden Sie die Grenzen der in dieser Lektion beschriebenen Instruktionen. Und schreiben Sie einige Programme alleine. Machen Sie Fehler - sie kosten nichts.

Zusammenfassung und Überblick

In dieser Lektion wurde beschrieben, wie ein Computer Schleifen ausführt und Entscheidungen fällt (d.h. Informationen verarbeitet). Mit Schleifen wird ein Verfahren mehrmals wiederholt. Anstatt dieselbe Zeile in einem Programm immer wieder zu wiederholen, können die Zeilen mit GOTO wiederholt werden. Dadurch wird beim Aufbau der Programme viel Zeit gespart.

Die IF...THEN Instruktion ist der "Entscheidungsfäller" des Computers. Mit IF...THEN können Sie zu verschiedenen Optionen verzweigen und Endlosschleifen beenden. Fehler können mit IF...THEN abgefangen werden, indem verhindert wird, daß der Benutzer Ihres Programms Informationen für INPUT-Instruktionen außerhalb des gültigen Bereichs eingibt.

Für das Schreiben von GOTO-Instruktionen innerhalb von IF...THEN Instruktionen wurden Abkürzungen gezeigt. Außerdem wurde beschrieben, wie das Fragezeichen anstelle von PRINT benutzt wird.

Schließlich haben Sie gesehen, wie Sie sich mit REM selbst an bestimmte Funktionen des Programms erinnern können. Durch Benutzung von REM in den Programmen können Sie die Programmzeilen besser organisieren. Mit REM werden die Programmsegmente so markiert, daß sie einfacher gefunden werden können. Somit wird das Austesten einfacher gestaltet.

Lektion 6

Grafik

Bis jetzt haben Sie auf Ihrem Apple Computer nur Text gesehen. Sie können jedoch auch einige wunderschöne Farbgrafiken erstellen. Der Computer verfügt über verschiedene Grafik-Betriebsarten. In dieser Lektion wird eine dieser Betriebsarten beschrieben, die **niedrig auflösende Grafik**. (Sie ist am einfachsten zu benutzen.)

Sie werden den Unterschied zwischen dem Text- und Grafik-Modus des Computers kennenlernen, während Sie sich mit den GR- und TEXT-Instruktionen befassen. Sie werden sehen, wie COLOR= benutzt wird, um eine von 16 Farben zur Benutzung mit PLOT (für das Zeichnen von Punkten), VLIN (für das Zeichnen vertikaler Linien) und HLIN (für das Zeichnen horizontaler Linien) festzulegen.

Darüber hinaus lernen Sie die RND-Instruktion für das Erzeugen von Zufallszahlen kennen - mit denen wiederum einige recht interessante Grafiken erstellt werden können.

Text und Grafik

Der Computer verfügt über verschiedene **Betriebsarten** für Text und Grafik. (Eine Betriebsart oder ein **Modus** ist eine von verschiedenen Möglichkeiten, mit denen der Computer Informationen interpretiert.)

Als erstes müssen Sie zwei Instruktionen kennenlernen - eine, um in den Grafik-Modus zu gehen und eine, um ihn zu verlassen. Beim Einschalten des Computers geht dieser automatisch in den Text-Modus. Wird die Instruktion **GR** für den Grafik-Modus eingegeben, so geht der Computer in den Grafik-Modus.

Geben Sie den Befehl

```
GR
```

(ohne Zeilennummer) ein und drücken Sie Return.

Der Bildschirm wurde gelöscht und der Cursor sprang zum Ende des Bildschirms. Der obere Teil des Bildschirms, über dem Cursor, ist für die Grafik reserviert. Sie belegt 20 der insgesamt 24 Zeilen des Bildschirms. Die vier letzten Zeilen sind für Texte reserviert.

❖ *Für Benutzer von Schwarz/Weiß-Monitoren:* In dieser Lektion wird davon ausgegangen, daß Sie einen Farb-Monitor oder ein Farbfernsehgerät benutzen. Benutzen Sie einen Schwarz/Weiß-Fernsehempfänger oder -Monitor, so werden die gezeichneten "Farben" durch unterschiedliche Graustufen oder Muster dargestellt.

Nun geben Sie folgendes Programm ein und führen Sie es aus:

```
NEW
10 GR                                                   Grafik-Modus ein.
20 COLOR= 3                                             Wählt eine Farbe aus.
30 PLOT 19, 19                                          Zeichnet einen Punkt.
100 PRINT "Rotes Quadrat in schwarzem Feld (1986)"      Berühmte Kunstwerke
                                                        haben immer einen Titel.
```

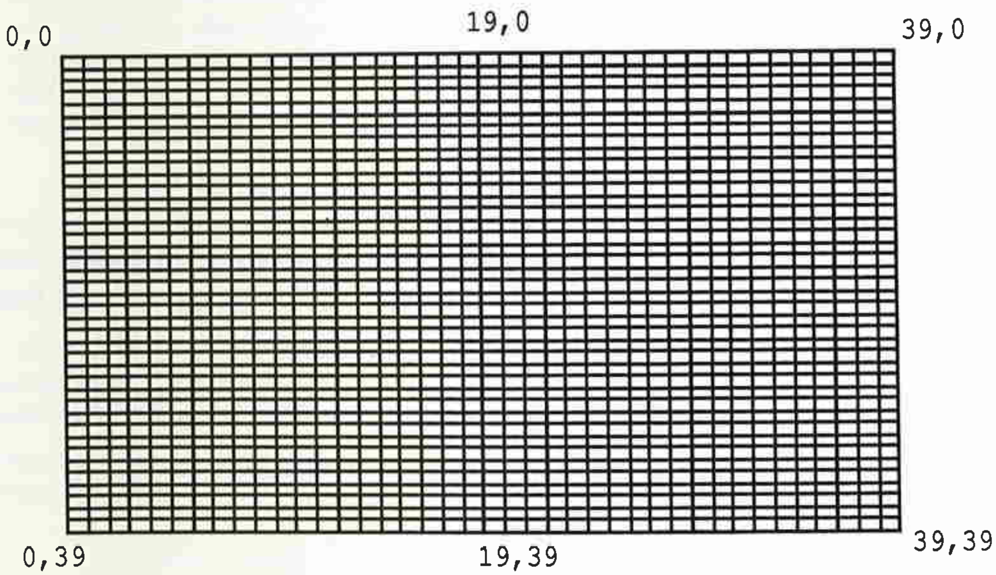
Eine 40 x 40 Leinwand

Der niedrigauflösende Grafik-Bildschirm entspricht einem 40 x 40 Raster. Mit der PLOT-Instruktion wird ein Punkt in die angegebene Horizontal- und Vertikalposition gezeichnet. Mit PLOT 0,0 wird er in die obere linke Ecke gezeichnet, während er mit PLOT 39,39 in die untere rechte Ecke gezeichnet wird.

Fügen Sie die folgenden Zeilen zu dem gerade eingegebenen Programm hinzu, um die Grenzen von PLOT kennenzulernen. (Machen Sie sich keine Gedanken, daß Sie den Rest des Programms nicht sehen können; in ein paar Minuten ist er wieder sichtbar.)

```
40 PLOT 0, 0
50 PLOT 39, 39
RUN
```

Wird das Programm nun ausgeführt, so sehen Sie drei Kästchen, die diagonal über den Bildschirm verteilt sind. Der Block oben links entspricht der Position 0,0 und der Block unten rechts der Position 39,39. Die ganze Matrix sieht folgendermaßen aus:



Bevor wir fortfahren, probieren Sie, ob Sie auch Punkte in die untere linke Ecke, die obere rechte Ecke und genau in die Mitte an den beiden Seiten zeichnen können. Wenn Sie erst einmal wissen, wie dies geht, können Sie an jeder Stelle zeichnen.

Erneute Anzeige der Auflistung

Nachdem neue Zeilen zu dem Programm hinzugefügt wurden, wurde der ganze Text über den neuen Zeilen hinter die Grafik gerollt und ist nicht mehr sichtbar. Um die Auflistung wieder anzuzeigen, muß wieder in den Text-Modus gegangen werden.

Mit der sofortigen Ausführung (d.h. mit einer Instruktion ohne Zeilennummer) geben Sie

```
TEXT
```

ein und drücken Return.

Das nun gezeigte seltsame Muster ist darauf zurückzuführen, daß der Apple nun seine eigenen Grafik-Symbole als Text interpretiert. Für Menschen sieht dies wie Unsinn (oder Kunst der "Neuen Wilden") aus. Geben Sie

```
HOME
```

ein und drücken Sie Return. Danach listen Sie das Programm auf. Haben Sie das letzte Programm richtig eingegeben, so sieht die Auflistung etwa folgendermaßen aus:

```
LIST
10 GR
20 COLOR= 3
30 PLOT 19, 19
40 PLOT 0, 0
50 PLOT 39, 39
60 PLOT 0, 39
70 PLOT 39, 0
80 PLOT 0, 19
90 PLOT 39, 19
100 PRINT "Rotes Quadrat in schwarzem Feld (1986)"
```

Farben mit COLOR= zeichnen

Mit der COLOR= Instruktion (das = ist Bestandteil der Instruktion) kann entschieden werden, welche Farben benutzt werden.

Nachfolgend eine Tabelle der verfügbaren Farben:

Nummer	Farbe	Nummer	Farbe
0	Schwarz	8	Braun
1	Magenta	9	Orange
2	Dunkelblau	10	Dunkelgrau
3	Rot	11	Rosa
4	Dunkelgrün	12	Grün
5	Grau	13	Gelb
6	Blau	14	Kobaltblau
7	Hellblau	15	Weiß

Durch die COLOR= Instruktion allein wird keine Farbe hinzugefügt. Nur das, was auf dem Bildschirm neu gezeichnet wird, wird in der neu gewählten Farbe dargestellt. Die mit COLOR= festgelegte Farbe bleibt bis zur nächsten COLOR= Instruktion wirksam.

Fügen Sie folgende neue Zeile zu dem Programm hinzu:

```
65 COLOR= 13
```

Nun führen Sie das Programm aus und sehen was geschieht.

❖ *Sauberer Text:* Wird im Grafik-Modus gearbeitet, so stehen nur vier Textzeilen zur Verfügung. Eine dieser vier Zeilen braucht nicht mit einer übriggebliebenen RUN-Instruktion verschandelt zu werden. Die Ästhetik ist immerhin auch ein Gesichtspunkt. Durch Hinzufügen einer HOME-Instruktion am Anfang des Programms (beispielsweise in Zeile 5) wird das Problem zufriedenstellend gelöst.

Benutzung von Variablen für das Zeichnen und die Angabe von Farben

Für das Zeichnen von Punkten und die Angabe von Farben können Variablen benutzt werden. Anstatt absolute Zahlen zu benutzen, wie bei COLOR= 10 oder PLOT10, 20 kann COLOR= FARBE oder PLOT SPALTE, ZEILE eingegeben werden.

Geben Sie das nächste Programm ein. Bevor Sie es ausführen, überlegen Sie sich, was geschehen wird:

```
NEW
10 GR
20 COLOR= 11                    11 ist rosa
30 PLOT SPALTE, ZEILE           Der Ausgangswert aller Variablen ist Null.
40 SPALTE = SPALTE + 1          Kein Grund zur Panik, die Erklärung folgt.
50 IF SPALTE > 39 GOTO 80       39 ist die höchste Spaltennummer im Raster.
60 ZEILE = ZEILE + 1
70 GOTO 30                      Und alles noch einmal.
80 END
```

Erhöhen von Spalten- und Zeilennummern

Zeile 40 wird in der Computerterminologie als Zähler bezeichnet. Wann immer der Computer Zeile 40 ausführt, wird der Wert des Zählers (namens SPALTE) um Eins erhöht. In der Umgangssprache bedeutet diese Zeile "Zu dem alten Wert von SPALTE den Wert 1 addieren. Ab jetzt den neuen Wert benutzen."

Der Originalwert von SPALTE ist 0 (sämtliche Variablen beginnen mit einem Wert von 0). Nachdem der Computer das erste Mal über Zeile 40 ging, steht 1 in SPALTE. Beim zweiten Mal steht 2 in SPALTE. Und so weiter, bis in SPALTE ein größerer Wert als 39 steht (gemäß Zeile 50) und das Programm endet.

❖ *Nur für Computer-Genies:* Zeichnen Sie eine diagonale Linie, die die erste Linie kreuzt - d.h. eine Linie, die in der oberen rechten Ecke beginnt und zur unteren linken Ecke geht. Dies ist schwieriger, als es sich anhört. Sobald Sie es jedoch herausgefunden haben, werden Sie erstaunt sein, wie einfach es Ihnen fällt.

Vielleicht.

Hinweis: Beginnen Sie bei 39 und arbeiten Sie rückwärts.

Zeichnen von horizontalen und vertikalen Linien

Mit der PLOT-Instruktion wird jeweils ein Punkt gezeichnet. Um mit PLOT eine vertikale oder horizontale Linie zu zeichnen, könnten Sie eine Folge von miteinander verbundenen Punkten programmieren, wie Sie es bei der diagonalen Linie getan haben. Bei Applesoft BASIC ist es jedoch wesentlich einfacher, HLIN (für *horizontale Linie*) und VLIN (für *vertikale Linie*) zu verwenden. Sie benutzen dieselben Zeichen-Koordinaten wie bei PLOT. Bei HLIN setzen Sie die *horizontale* Anfangs- und Endposition mit der AT-Instruktion in eine vertikale Position:

```
HLIN ANFANG, ENDE AT ZEILE
```

Bei VLIN geben Sie die *vertikale* Anfangs- und Endposition in einer horizontalen Position an:

```
VLIN ANFANG, ENDE AT SPALTE
```

Betrachten Sie das nächste Beispiel, um zu sehen, wie ein Kreuz auf dem Bildschirm gezeichnet wird:

```
10 GR
20 COLOR= 15
30 HLIN 10, 30 AT 19             Malt Linie von links nach rechts.
40 VLIN 10, 30 AT 19             Malt Linie nach oben und unten.
```

Die Zeilen 30 und 40 sehen identisch aus, nur wird in der einen HLIN und in der anderen VLIN benutzt.

Als Übung ändern Sie Zeile 30 so, daß die Linien anstelle eines Kreuzes ein T darstellen, wobei die horizontale Linie direkt über dem Ende der vertikalen Linie liegt.

Ein allgemeines Programm für das Zeichnen von Linien

Mit diesem Programm können Sie verschiedene Werte eingeben, um unterschiedliche Linien zu zeichnen. Benutzen Sie dieses Programm, bis Sie ein Gefühl dafür gewonnen haben, wo verschiedene Werte die Linien in der Matrix zeichnen:

```
NEW
5 TEXT
10 HOME
20 INPUT "Erster Block von HLIN: "; HB
30 INPUT "Letzer Block von HLIN: "; HE
40 INPUT "Zeile für HLIN: "; HP
50 INPUT "Erster Block von VLIN: "; VB
60 INPUT "Letzter Block von VLIN: "; VE
70 INPUT "Spalte für VLIN: "; VP
100 REM ************************
110 REM ZEICHNET LINIEN
120 REM ************************
130 GR
140 COLOR= 15
150 HLIN HB, HE AT HP
160 VLIN VB, VE AT VP
170 INPUT "Weitere Zeilen (J/N)? "; AN$
180 IF AN$ = "J" THEN 10
```

Experimentieren Sie mit verschiedenen Werten, bis Sie genau vorhersagen können, wo die vertikalen und horizontalen Linien verlaufen. Zur Übung geben Sie Werte ein, die außerhalb des Matrixbereichs liegen (d.h. größer sind als 39). Geben Sie beispielsweise einen Wert von 50 ein, und sehen Sie, was geschieht. Es ist genauso wichtig, die Bedeutung von Fehlermeldungen zu verstehen, wie gelernt werden muß, Funktionen auszuführen, ohne Fehlermeldungen zu erhalten. Begehen Sie dann später einen Fehler (und jeder macht Fehler beim Lernen), so haben Sie eine bessere Vorstellung, wie Sie den Fehler beheben können.

Bevor Sie fortfahren, ändern Sie das Programm so, daß es Sie fragt, welche Farbe benutzt werden soll. Fühlen Sie sich wirklich auf der sicheren Seite, so schreiben Sie schnell noch ein paar Zeilen, mit denen die Zeichenkoordinaten am unteren Rand des Bildschirms angezeigt werden. Der angezeigte Text sollte folgendermaßen aussehen:

```
Horizontale Linie von 10 bis 35 in Zeile 15

Vertikale Linie von 18 bis 26 in Spalte 25
```

Beliebige Grafik

In den Computer ist ein Generator für Zufallszahlen eingebaut. Mit ihm kann der Computer Zahlen aus dem Hut zaubern. Mit der RND-Instruktion selbst werden Zufalls-Dezimalzahlen zwischen 0 und 1 generiert.

Versuchen Sie folgendes Programm:

```
NEW
10 TEXT                          Das letzte Programm hat Grafik benutzt.
20 HOME                          Alte Zeichen werden gelöscht.
30 COUNT = COUNT + 1             Zu dem Zähler wird 1 addiert.
40 PRINT RND (1)                 Eine Zufallszahl wird ausgedruckt.
50 IF COUNT = 5 THEN GOTO 70     Schon 5 Zahlen gedruckt?
60 GOTO 30                       Wenn nicht, hole eine andere Zufallszahl.
70 END                           Wenn ja, wird das Programm beendet.
```

Mit RND wird immer eine Dezimalzahl zwischen 0 und 1 gedruckt. Durch Multiplikation des Ergebnisses mit einer Ganzzahl können jedoch Zahlen erstellt werden, mit denen der Computer Grafiken zeichnet. Ändern Sie Zeile 40 folgendermaßen:

```
40 PRINT RND (1) * 40            Die Klammern sind obligatorisch.
```

Führen Sie das Programm erneut aus. Alle Zahlen sind größer als 0 und kleiner als 40.

❖ *Obligatorische Klammern mit RND:* Auf RND muß eine Zahl in Klammern folgen. Um zu gewährleisten, daß RND bei jeder Benutzung eine andere Reihe von Zufallszahlen erzeugt, muß 1 oder eine höhere Zahl angegeben werden. (Für Experimentierfreudige: Um eine sich wiederholende Folge von Zahlen zu erhalten, benutzen Sie 0 oder eine negative Zahl.)

Geben Sie die folgende Variation desselben Programms ein und führen Sie sie aus. Mit ihr wird jede Zufallszahl in eine Variable gesetzt, während die Zufallszahl erzeugt wird:

```
5 TEXT
10 HOME
20 NUMBER = RND (1) * 40
30 PRINT NUMBER
40 IF NUMBER > 38 THEN GOTO 60
50 GOTO 20
60 PRINT "Das war´s!"
70 END
```

Dieses Programm wird ausgeführt, bis der Zufallszahlen Generator eine Zahl erzeugt, die größer ist als 38. Gelegentlich werden sehr viele Zahlen aufgelistet und dann wieder nur ganz wenige, je nachdem, wie schnell eine Zahl erzeugt wird, die größer ist als 38. Beachten Sie übrigens, daß das Programm Zahlen zwischen 0 und 39,9999 generiert - niemals eine Zahl, die 40 erreicht.

Zufallsgrafiken werden erzeugt, indem zufällig generierte Variablen in PLOT benutzt werden. Sie können auch Zufallszahlen benutzen, um verschiedene Farben zu erzeugen.

Für einige besonders farbige Ergebnisse geben Sie das nächste Programm ein und führen es aus:

```
10 GR
15 REM FARBEN 0 - 15
20 FARBE = RND (1) * 16
25 REM HORIZONTALWERTE 0 - 39
30 SPALTE = RND (1) * 40
35 REM VERTIKALWERTE 0 - 39
40 ZEILE = RND (1) * 40
50 COLOR= FARBE
60 PLOT SPALTE, ZEILE
70 IF ZEILE > 39 THEN END
80 GOTO 20
```

❖ *Was geschieht mit dem Bruchteil?* Eine Grafik-Instruktion betrachtet nur den ganzzahligen Teil einer Zahl. Der Bruchteil wird ignoriert. Für eine Grafik-Instruktion ist 39,999999 gleichbedeutend mit 39. 1,111111 ist gleichbedeutend mit 1. Jede positive Zahl, die kleiner ist als 1, ist gleich 0.

Eine kleine Herausforderung für Sie: Nichts schwieriges - ändern Sie das Programm einfach so, daß es zufällig horizontale und vertikale Linien mit zufälliger Länge generiert.

Zusammenfassung und Überblick

Farbgrafiken erweitern die Programmierung um eine weitere Dimension. Mit ihnen können Sie nützliche Programme erstellen, und es macht viel Spaß, mit ihnen zu arbeiten. Bei der niedrigauflösenden Grafik werden keine ausgefeilten Zeichnungen erstellt, sie haben jedoch viel Farbe und stellen gute Grafiken dar. Sie können PLOT, HLIN, VLIN und COLOR= mit anderen Programmier-Instruktionen benutzen, um Grafiken zu erstellen. Mit dem Zufallszahlen-Generator in dem Apple können beliebige Zahlenbereiche ausgegeben werden. Werden RND und die Grafik-Instruktionen kombiniert, so kann ein ganzes Kaleidoskop an Formen und Farben erstellt werden.

Lektion 7

Kontrollierte Schleifen

In dieser Lektion werden die Schleifen noch näher beschrieben. Aus den vorherigen Lektionen kennen Sie schon Schleifen mit GOTO. In dieser Lektion wird die FOR\NEXT Instruktion erläutert, mit der im voraus entschieden werden kann, wie oft eine Schleife ausgeführt wird. Sie werden einige Tricks bei der Benutzung von Schleifen kennenlernen (beispielsweise das Verlangsamen der Programmausführung). Und schließlich werden Sie noch lernen, wie einfache bewegte Grafik erstellt wird.

Die Lektion wird mit einer Liste sämtlicher Befehle, Instruktionen, Operatoren und Programmier-Grundlagen beendet, die Sie bis jetzt erlernt haben. Diese Liste wird Sie beeindrucken.

FOR\NEXT

In Lektion 5 wurde beschrieben, wie mit einem Zähler mit IF...THEN festgelegt wurde, wie oft der Computer eine Schleife ausführt:

```
NEW
10 GR
20 COLOR= 11
30 PLOT SPALTE, ZEILE                  Die Schleife beginnt hier.
40 SPALTE = SPALTE  + 1                Dies ist der Zähler.
50 IF SPALTE > 39 GOTO 80              um Schleife zu verlassen.
60 ZEILE = ZEILE + 1
70 GOTO 30                             Hier endet die Schleife.
80 END
```

Mit der FOR\NEXT Instruktion kann zu Beginn definiert werden, wie viele Schleifen das Programm ausführt. Sie verfügt über einen eingebauten eigenen Zähler. Nachfolgend die Struktur dieser zweiteiligen Instruktion:

FOR <*Variable* > = <*Start* > TO <*Ende* >

<Die hier stehenden Instruktionen werden ausgeführt>

NEXT <*Variable* >

Das folgende Programm benutzt FOR\NEXT, um eine Schleife zehnmal zu wiederholen. Geben Sie das Programm ein und führen Sie es aus:

```
NEW
10 TEXT                              Das letzte Programm war ein Grafik-
                                     Programm; stellt den Textmodus wieder her
20 HOME                              Löscht nicht mehr benötigte Zeichen.
30 FOR RUNDE= 1 TO 10                Dies ist der FOR-Teil...
40 PRINT "Dies ist Runde # ", RUNDE  alle Instruktionen innerhalb der Schleife
                                     werden ausgeführt...
50 NEXT RUNDE                        dies ist der NEXT-Teil.
```

Wird dieses Programm ausgeführt, so geht der Wert von RUNDE von 1 auf 10. Die Variable RUNDE verhält sich wie jede andere Variable. Wie aus der Bildschirmanzeige hervorgeht, stellen die Zahlen die Werte dar, die die Schleife generiert. Sämtliche Zeilen zwischen FOR und NEXT werden wiederholt, bis die Schleife ihren Höchstwert erreicht. In diesem Fall ist der Wert gleich 10.

Die Schleife kann mit jedem beliebigen Wert gestartet werden. Nachfolgend verschiedene Versionen der Zeile 30, die (natürlich nacheinander) geändert werden können, um zu sehen, was geschieht:

```
30 FOR RUNDE = 0 TO 20
30 FOR RUNDE = -10 TO 10             Beginn mit einer negativen Zahl.
30 FOR RUNDE = 128 TO 255
```

Anstelle von Zahlen für die FOR\NEXT Schleife können auch Variablen benutzt werden. Mit dem folgenden Programm kann beispielsweise INPUT benutzt werden, um die Anfangs- und Endwerte der Schleife festzulegen:

```
NEW
10 HOME
20 INPUT "Niedrigste Zahl: "; NIEDRIG
30 INPUT "Hoechste Zahl: "; HOCH
40 HOME
50 FOR NUM = NIEDRIG TO HOCH
60 PRINT NUM
70 NEXT NUM
```

Die FOR\NEXT Schleife kann auch mit Grafiken gut benutzt werden. Mit einer FOR\NEXT Schleife können diagonale Linien zu den schon vorhandenen vertikalen und horizontalen Linien gezeichnet werden. Hier das Originalprogramm:

```
10 GR
20 COLOR= 11
30 PLOT SPALTE, ZEILE                Die Schleife beginnt hier.
40 SPALTE = SPALTE + 1               Hier ist der Zähler...
50 IF SPALTE > 39 GOTO 80            ...um die Schleife zu verlassen.
60 ZEILE = ZEILE + 1
70 GOTO 30                           Die Schleife endet hier.
80 END
```

Nachfolgend die FOR\NEXT Version:

```
10 GR
20 COLOR= 11
30 FOR ZAEHLER = 0 TO 39
40 PLOT ZAEHLER, ZAEHLER
50 NEXT ZAEHLER
```

Benutzung von STEP mit FOR\NEXT

Gelegentlich wird man rückwärts zählen oder Zahlen in einem Programm überspringen. Benutzen Sie STEP mit FOR\NEXT, um anzugeben, in welche Richtung gezählt und welche Erhöhung benutzt wird.

Bei diesem Programm wird beispielsweise in Fünfer-Schritten gezählt. Geben Sie das Programm ein und führen Sie es aus:

```
10 HOME
20 FOR ZAHL = 10 TO 100 STEP 5
30 PRINT ZAHL
40 NEXT ZAHL
```

Und mit diesem Programm wird rückwärts gezählt:

```
10 HOME
20 FOR COUNTDOWN = 10 TO 0 STEP -1
30 PRINT COUNTDOWN
40 NEXT COUNTDOWN
50 PRINT "START!"
```

(Fünf zusätzliche Punkte, wenn Sie die Rakete zeichnen können.)

Sie können sogar einfache bewegte Grafik zeichnen, bei denen sich ein Block auf dem Bildschirm bewegt. Hier ist es ein Ball, der hin- und herspringt:

```
NEW
10 GR
20 FOR SPRING = 0 TO 39
30 COLOR = 15                          Legt die Farbe auf Weiß fest...
40 PLOT 19, SPRING                     ...damit Sie den Ball sehen können.
50 COLOR = 0                           Farbe wird jetzt schwarz...
60 PLOT 19, SPRING                     ...damit Sie ihn löschen können.
70 NEXT SPRING
100 REM*************
110 REM SPRING HOCH
120 REM *************
130 FOR SPRING = 39 TO 0 STEP -1
140 COLOR = 15
150 PLOT 19, SPRING
160 COLOR = 0
170 PLOT 19, SPRING
180 NEXT SPRING
```

Sie sehen , wie einfach dies mit einem Rückwärts-STEP ist. Der Ball springt übrigens länger, wenn Sie

```
190 GOTO 20
```

hinzufügen. Richtig hübsch wird es, wenn die Zeilen 15 und 140

```
COLOR = RND ( 1) * 16
```

lauten. Damit der Ball diagonal springt, ändern Sie... - eigentlich sollten Sie es selbst herausfinden.

Verzögerungsschleifen

Gelegentlich wird man das Programm verlangsamen wollen, damit man die Dinge auf dem Bildschirm verfolgen kann, die normalerweise zu schnell gehen.

Geben Sie beispielsweise das nächste Programm ein und führen Sie es aus, um eine Meldung auf dem Bildschirm anzuzeigen, den Bildschirm zu löschen und eine andere Meldung anzuzeigen:

```
10 HOME
15 DAUER = 1000
20 PRINT "EINE WICHTIGE MELDUNG"                 Meldung anzeigen...
30 FOR PAUSE=1 TO DAUER                          ...festhalten...
35 NEXT PAUSE
40 HOME                                          ...Bildschirm löschen.
50 PRINT "PROGRAMM BITTE SICHERN"                Meldung anzeigen...
60 FOR PAUSE=1 TO DAUER                          ...festhalten...
65 NEXT PAUSE
70 HOME                                          ...Bildschirm löschen.
80 PRINT "BEVOR SIE DEN COMPUTER AUSSCHALTEN!"
```

Durch die leeren FOR\NEXT Schleifen zwischen der Anzeige der Meldungen und den HOME-Instruktionen haben Sie Zeit, den Bildschirminhalt zu lesen. (Nehmen Sie die Zeilen 30, 35, 60 und 65 heraus- geben Sie einfach die Zeilennummern ein und drücken Sie Return - und die Meldungen werden zu schnell ausgegeben, um während der Ausführung des Programms gelesen werden zu können.)

Verzögerungsschleifen werden benutzt, wenn verschiedene Meldungen automatisch angezeigt und keine Tasten gedrückt werden sollen, um die nächste Meldung anzuzeigen. Mit kurzen Verzögerungsschleifen können Programme mit Blitzlichteffekten erstellt werden.

Für einen Rechtschreibungs-Quiz wird ein Wort lang genug auf dem Bildschirm angezeigt, um gelesen werden zu können, jedoch nicht lange genug, um buchstabiert werden zu können. Hier ein kurzer Versuch:

```
NEW
5 DAUER = 150                                    Dieser Wert ist die Pausenlänge.
10 HOME
20 REM *****************
30 REM WOERTER BUCHSTABIEREN
40 REM *****************
50 A$ = "ENTE"
60 B$ = "SCHMUCK"
70 C$ = "PROGRAMMIERUNG"
100 REM ****************
110 REM RECHTSCHREIBPRUEFUNG
120 REM ****************
130 PRINT A$
140 FOR LOOK = 1 TO DAUER                        Verzögerungsschleife.
145 NEXT LOOK
150 HOME
160 INPUT "WORT BUCHSTABIEREN "; BUCHST$
170 IF BUCHST$ = A$ THEN OK = OK + 1             Zähler addiert richtige Schreibweise.
180 PRINT B$
190 FOR LOOK = 1 TO DAUER                        Verzögerungsschleife.
195 NEXT LOOK
200 HOME
210 INPUT "WORT BUCHSTABIEREN "; BUCHST$
220 IF BUCHST$ = B$ THEN OK = OK + 1
230 PRINT C$
240 FOR LOOK = 1 TO DAUER                        Verzögerungsschleife.
245 NEXT LOOK
250 HOME
260 INPUT "WORT BUCHSTABIEREN "; BUCHST$
270 IF BUCHST$ = C$ THEN OK = OK + 1
280 HOME
290 PRINT "Sie haben "; OK; " Worte richtig buchstabiert."
```

Sie können den Wert für die Verzögerungsschleife (Zeile 5) ändern, um mehr oder weniger Zeit für die Betrachtung des Wortes zu haben.

Ein kurzer Überblick

Sie sind mit der Programmierung schon recht fortgeschritten. Deshalb hier ein kurzer Überblick über die Schritte, die Sie in den sieben ersten Lektionen erlernt haben. Generell ist es wichtig, die Dinge so einfach wie möglich zu halten - die Programmierung sollte immer auf kleine Mengen beschränkt sein. Nachfolgend eine Liste der bis jetzt erlernten Funktionen. Haben Sie einen dieser Ausdrücke vergessen, so schlagen Sie in dem Glossar nach oder gehen zu der entsprechenden Lektion zurück, um diesen Teil noch einmal zu lesen.

Befehle

CAT	DELETE	NEW
LIST	LOAD	PR#1
PR#0	RUN	SAVE

Instruktionen

COLOR=	END	GR
HOME	FOR [STEP]\NEXT	GOTO
HLIN	IF...THEN	INPUT
PLOT	PRINT	RND
REM	TEXT	VLIN

Operatoren

+	*	
/	(	)
<	>	=
>=	<=	

Grundlagen

Aussagefähige Namen	Bedienerhinweise
Numerische Variablen	Priorität
Sofortige und verzögerte Ausführung	
Schleifen	Verzögerungsschleifen
Zähler	Zeichenfolgen-Variablen
Zeilennummern in Zehnerschritten	

Zeit zum Üben

In den nächsten drei Lektionen werden die schon erlernten Instruktionen und Techniken etwas verfeinert. Außerdem werden weitere Tricks und Techniken vorgestellt. Bevor Sie fortfahren, sollten Sie Ihre bis jetzt erworbenen Kenntnisse nutzen, um eigene Programme zu schreiben und etwas zu experimentieren. Sie müssen unbedingt Spaß an Ihrer Arbeit mit dem Computer haben. Schreiben Sie Programme, die Ihnen Spaß machen, so lernen Sie nicht nur schneller programmieren, sondern haben dabei auch noch Freude.

Zusammenfassung und Überblick

In dieser Lektion wurde noch einmal mit Schleifen gearbeitet - dies waren jedoch kontrollierte Schleifen. Die Benutzung von Zählern wurde verfeinert und eine neue Schleife namens FOR\NEXT vorgestellt. Sie haben ein wenig über bewegte Grafik gelernt und gesehen, wie ein Programm mit Verzögerungsschleifen verlangsamt werden kann. Danach haben Sie sämtliche bis jetzt erlernten Instruktionen und Grundlagen noch einmal durchgearbeitet und neue, eigene Programme erstellt.

Lektion 8

Programmieren mit Stil: Modulare Programmierung

Nun verfügen Sie über genügend Kenntnisse, um eigene Programme zu schreiben. Schließlich erstellen Sie ein Programm, mit dem Ihr Kontostand berechnet wird.

Beachten Sie das Wort "Erstellen" im letzten Satz. Die besten Programme sind keine puren Auflistungen von Programmzeilen. Im Gegenteil, sie stellen gut durchdachte Sammlungen von Programmsegmenten dar, wobei jedes Segment seine eigene Aufgabe ausführt. In dieser Lektion werden die Grundlagen der Programmorganisation und der Programm-Moduln erläutert.

GOSUB\RETURN

In verschiedenen Teilen eines Programms werden Sie häufig dieselbe Funktion ausführen wollen. So haben Sie beispielsweise in Lektion 7 dieselbe Verzögerungsschleife dreimal in einem recht kurzen Programm benutzt:

```
60 FOR PAUSE = 1 TO DAUER
70 NEXT PAUSE
```

Stellen Sie sich nun ein Programm vor, in dem Sie dieselben Zeilen 10, 20 oder sogar 30-mal benutzen - und wie mühsam es ist, dieselbe Zeile wieder und wieder einzugeben (und wieviel Speicherplatz das Programm im RAM belegt). Nun wollen wir eine Situation betrachten, die recht häufig auftritt, bei der die wiederholte Routine (d.h. die Sammlung von Zeilen zur Ausführung einer bestimmten Funktion) nicht nur vier Zeilen umfaßt, sondern 10 oder mehr Zeilen. Bis dies fertig ist, werden Sie eine Hornhaut an Ihren Fingerspitzen haben.

Für diese Situationen ist die GOSUB\RETURN Instruktion von BASIC gedacht. Sie geben eine Routine nur einmal ein und benutzen dieselben Zeilen (mit genau denselben Zeilennummern) wieder und wieder.

Hier das Pause-Programm ohne GOSUB\RETURN:

```
5 DAUER = 1000
10 HOME
20 PRINT "EINE WICHTIGE MELDUNG"
30 FOR PAUSE=1 TO DAUER
35 NEXT PAUSE
40 HOME
50 PRINT "PROGRAMM BITTE SICHERN"
60 FOR PAUSE = 1 TO DAUER
65 NEXT PAUSE
70 HOME
80 PRINT "BEVOR SIE DEN COMPUTER AUSSCHALTEN!"
```

Und hier dasselbe Programm *mit* GOSUB\RETURN. Geben Sie es ein und starten es:

```
5 DAUER = 1000
10 HOME
20 MELDUNG$ = "EINE WICHTIGE MELDUNG"
30 GOSUB 210                          Gehe zu einem Unterprogramm in Zeile 210
40 MELDUNG$ = "PROGRAMM BITTE SICHERN"
50 GOSUB 210
60 MELDUNG$ = "BEVOR SIE DEN COMPUTER AUSSCHALTEN!"
70 GOSUB 210
190 END                               Dies muß hier stehen.
200 REM **** UNTERPROGRAMM FUER MELDUNG*****
210 HOME                              Unterprogramm beginnt hier.
220 PRINT MELDUNG$
230 FOR PAUSE = 1 TO DAUER
240 NEXT PAUSE
250 RETURN                            Hier endet das Unterprogramm; das
                                      Programm kehrt an die Stelle zurück, von
                                      der es mit GOSUB her geschickt wurde.
```

GOSUB bedeutet "Gehe zum Unterprogramm". (Ein **Unterprogramm** ist eine Routine innerhalb eines Programms, zu der mit einer GOSUB-Instruktion gegangen wird. GOSUB kommt vom amerikanischen Wort subroutine.) Wie bei GOTO verläßt das Programm bei GOSUB die normale Folge der Zeilennummern, um eine bestimmte Funktion auszuführen. Im Gegensatz zu GOTO kehrt GOSUB jedoch wieder an die ursprüngliche Stelle zurück. Dies erfolgt mit RETURN am Ende des Unterprogramms. Sie brauchen sich die Zeilennummer, zu der zurückgekehrt werden soll, nicht zu merken. GOSUB\RETURN tut das für Sie.

END schützt Unterprogramme

Unterprogramme stehen im allgemeinen am Ende eines Programms, wie aus den Beispielen in dieser Lektion hervorgeht. Zwischen dem Hauptprogramm und den Unterprogrammen muß eine END-Instruktion eingefügt werden.

Um dies verständlich zu machen, nehmen Sie die END-Instruktion in Zeile 190 heraus und führen das Programm aus. (Um eine Zeile herauszunehmen, geben Sie einfach die Zeilennummer ein und drücken Return.)

Sie haben die Fehlermeldung RETURN WITHOUT GOSUB erhalten. Der Computer erwartet eine RETURN-Instruktion nur, wenn mit GOSUB zu einem Unterprogramm gegangen wird. Stößt der Computer versehentlich (wie in diesem Fall) auf eine RETURN-Instruktion, so weiß er nicht, wohin er zurückgehen soll, ist verwirrt und gibt dies mit einer Fehlermeldung an.

Eine Möglichkeit, um die Unterprogramme sicher vom Hauptprogramm zu trennen, besteht darin, von vornherein zu entscheiden, in welcher Zeilennummer die Unterprogramme starten. Danach wird eine Zeilennummer und eine END-Instruktion direkt vor diese Zeilennummer gesetzt. In dem Programm, mit dem gerade gearbeitet wurde, beginnt das Unterprogramm in Zeile 210, direkt hinter der REM-Instruktion in Zeile 200; deshalb wird die END-Instruktion in Zeile 190 eingefügt.

Unterprogramme und Organisation

Auch im nächsten Beispiel werden Unterprogramme verwendet, jedoch nicht, weil das Programm bestimmte Zeilensegmente häufig wiederholt, sondern weil das Programm auf diese Weise einfacher zu lesen und besser organisiert ist. Je größer Ihre Erfahrung als Programmierer wird, desto länger werden Ihre Programme und desto mehr Funktionen führen sie aus. Hier wird eine gute Organisation des Programms immer wichtiger.

Geben Sie das folgende Programm ein und führen Sie es aus. Beachten Sie die neuen Dinge in einigen Zeilen mit REM-Instruktionen:

```
5 REM *****************************************
10 REM Programm für Zufallszahlen
12 REM Dieses Programm generiert so viele Zufallszahlen
14 REM wie der Benutzer will. Mit ihm kann der Benutzer
16 REM auch den Zahlenbereich festlegen.
18 REM *****************************************
20 GOSUB 1010 : REM Titelblatt
30 GOSUB 1110 : REM Wie viele Zahlen und welcher Bereich?
40 GOSUB 1210 : REM Generiert Zufallszahlen
50 GOSUB 1310 : REM Noch einmal?
60 IF AN$ = "J" THEN 30 :   REM Wenn ja, wiederholen....
70 PRINT
80 PRINT "Vielen Dank." : REM wenn nicht, Ende
999 END
1002 REM *****************************************
1004 REM Titelblatt
1006 REM **************************************
1010 HOME
1020 PRINT "Zufallszahlengenerator"
1030 PRINT
1040 PRINT "Dies Programm druckt beliebige Zufallszahlen"
1050 PRINT "zwischen 0 und einem gewaehlten Grenzwert."
1060 PRINT
1070 PRINT
1080 INPUT "Return zum Starten:  "; Start$
1090 RETURN
```

```
1102 REM *******************************************
1104 REM Wie viele Zahlen und welche Grenze?
1106 REM *******************************************
1110 HOME
1120 INPUT "Wie viele Zahlen wollen Sie haben? "; RNUMS
1130 PRINT
1140 INPUT "Wie gross kann eine Zahl maximal sein? "; LIMIT
1150 RETURN
1202 REM *******************************************
1204 REM Generiert Zufallszahlen
1206 REM *******************************************
1210 FOR COUNT = 1 TO RNUMS
1220 NUM = RND ( 1) * LIMIT
1230 PRINT NUM
1240 NEXT COUNT
1250 PRINT
1260 RETURN
1302 REM *******************
1304 REM Noch einmal ?
1306 REM *******************
1310 INPUT "Noch mehr Zufallszahlen? (J/N) "; AN$
1320 RETURN
```

Dieses Programm benutzt viele Unterprogramme, mit denen der Ablauf einfacher verfolgt werden kann. Werden außerdem die REM-Instruktionen und aussagefähige Variablennamen benutzt, so haben Sie ein besonders leicht zu verfolgendes Programm - sowohl jetzt, nachdem Sie es gerade geschrieben haben, als auch sechs Monate später, wenn Sie einige Zeilen ändern möchten.

Mehrere Instruktionen in einer Zeile

Wahrscheinlich haben Sie schon festgestellt, daß Sie mehr als eine Instruktion auf eine Zeile setzen können, wenn Sie zwischen die Instruktionen einen Doppelpunkt (:) setzen. In dem vorhergehenden Programm gibt es viele Beispiele hierfür. In diesem Programm wird der Doppelpunkt nur benutzt, um REM-Instruktionen hinzuzufügen. Sie können den Doppelpunkt jedoch auch für alle anderen Instruktionen benutzen. Dennoch ist Vorsicht geboten. Gelegentlich können die Ergebnisse recht überraschend sein. Beginnen Sie beispielsweise eine Zeile mit einer REM-Instruktion, so ignoriert der Computer die ganze Zeile und nicht nur die REM-Instruktion:

```
20 REM Bemerkung:GOSUB 1010
```
GOSUB wird ignoriert!

Im Laufe der Zeit werden Sie noch manche derartige Überraschung erleben.

Entwurf Ihres Programms Schritt für Schritt

Häufig ist der Umfang eines Programms überwältigend. Es scheint zu komplex, zu lang oder einfach zu schwierig. Gelegentlich ist dies auch richtig. Sie können sicherlich noch kein Programm schreiben, mit dem der Bundeshaushalt verwaltet wird (das scheinen andere auch nicht zu können). Mit Ihren Kenntnissen können Sie jedoch schon mehr tun als Sie denken. So können Sie beispielsweise ein Programm schreiben, mit dem Ihr Kontostand berechnet wird.

Der Trick besteht darin, die Aufgabe in einfach zu verwaltende Segmente zu unterteilen. Überlegen Sie einen Augenblick, wie Sie Ihren Kontostand von Hand errechnen:

1. Der alte Kontostand wird festgehalten.
2. Gutschriften werden addiert.
   a. Der Betrag einer Gutschrift wird festgehalten.
   b. Er wird dem Saldo hinzugefügt, um den neuen Kontostand zu erhalten.
   c. Die Schritte a) und b) werden wiederholt, bis alle Gutschriften addiert wurden.
3. Die Beträge für Belastungen werden subtrahiert.
   a. Die Abbuchungen werden festgehalten.
   b. Sie werden vom Saldo subtrahiert, um einen neuen Saldo zu erhalten.
   c. Die Schritte a) und b) werden wiederholt, bis alles abgebucht ist.
4. Der Saldo wird ausgedruckt.

Damit wurde der **Algorithmus** (d.h. die Methode zur Problemlösung) für die Ermittlung des Kontostandes geschrieben. Als nächstes müssen Moduln für die einzelnen Schritte in dem Algorithmus geschrieben werden. Hierzu müssen die Moduln nur auf die richtige Art und Weise angeordnet werden. Beim Programmaufbau werden einfache Moduln nur so hintereinandergesetzt, daß sie gemeinsam benutzt werden können.

Das große Konto-Programm

Benutzen Sie den Algorithmus, um ein eigenes Programm zur Berechnung des Kontostandes zu schreiben. Fügen Sie ein Modul hinzu, mit dem ein kleines Menü erstellt wird, aus dem Sie die als erstes auszuführenden Aufgaben auswählen können - Addition der Abbuchungen oder Addition der Gutschriften.

Nachdem Sie Ihre eigene Programmversion geschrieben haben, lassen Sie sie ausdrucken. Danach vergleichen Sie Ihr Programm mit dem nachfolgend aufgeführten Programm. Betrachten Sie dies als Möglichkeit, zu sehen, ob Sie die Ausführungen in diesem Tutorium wirklich verstanden haben. Nehmen Sie sich Zeit und denken Sie daran, oft REM-Zeilen zu benutzen.

Eine Version des Konto-Programms

Dies ist nur eine Version. Kann Ihre Version erfolgreich ausgeführt werden, so ist sie genauso gut wie diese. Diese Version soll Ihnen hier nur weiterhelfen, falls Sie Probleme haben.

Der wichtigste Punkt dieser Version ist die Unterteilung der Aufgaben in kleine Schritte.

Modul 1

```
5 REM **************************
10 REM BERECHNUNG DES KONTOSTANDES
15 REM *************************
20 HOME
30 INPUT "Alten Kontostand eingeben: DM "; SALDO
40 PRINT
50 PRINT "1. Gutschriften eingeben"
60 PRINT "2. Belastungen eingeben"
70 PRINT "3. Ende"
80 INPUT "BITTE NUMMER ANGEBEN "; NUMMER
90 IF NUMMER = 1 THEN GOSUB 200
100 IF NUMMER = 2 THEN GOSUB 300
110 IF NUMMER = 3 THEN GOTO 170
120 IF NUMMER > 3 THEN GOTO 40
```

Außerhalb des Bereichs liegende Zahlen werden abgefangen.

```
130 PRINT "Laufender Kontostand DM "; SALDO
140 PRINT
150 INPUT "Return um fortzufahren: ";OK$
```

Wartet, bis der Benutzer bereit ist.

```
160 GOTO 40
170 PRINT
180 PRINT "Ihr Endsaldo ist DM "; SALDO
190 END
```

Das erste Modul stellt den "Rahmen" des Programms dar. Unterprogramme übernehmen die anderen Aufgaben. Mit dem nächsten Modul werden Gutschriften verarbeitet und zu dem Saldo hinzugefügt.

Modul 2

```
200 REM ************************
210 REM GUTSCHRIFTEN ADDIEREN
220 REM ************************
230 HOME
240 INPUT "Wie viele Gutschriften sind erfolgt? "; ND
250 FOR X= 1 TO ND
260 INPUT "Betrag der Gutschrift: DM ";GUT
270 SALDO = SALDO + GUT                    Ermittelt laufende Gesamtsumme
280 NEXT X
290 RETURN
```

Als nächstes tun Sie dasselbe für die Belastungen, anstatt allerdings zum Saldo zu addieren, wird vom Saldo subtrahiert.

Modul 3

```
300 REM *************************
310 REM BELASTUNGEN ANGEBEN
320 REM *************************
330 HOME
340 INPUT "Wie viele Belastungen wurden vorgenommen? "; NC
350 FOR X= 1 TO NC
360 INPUT "Betrag der Belastung: DM "; SCHECK
370 SALDO = SALDO - SCHECK                 Ermittelt laufende Gesamtsumme
380 NEXT X
390 RETURN
```

Wenn Sie das Programm ansehen, können Sie gut feststellen, was jeder Teil tut. Aus den REM-Zeilen wird leicht ersichtlich, was in den Unterprogrammen geschieht.

Sichern Sie das Programm auf Diskette. In der nächsten Lektion werden Sie lernen, wie man Programme attraktiver gestaltet, und dieses Programm ist gut zum Üben.

Zusammenfassung und Überblick

In dieser Lektion haben Sie die GOSUB\RETURN-Instruktion kennengelernt und gleichzeitig erfahren, wie wichtig ein guter Programmaufbau ist.

Mit GOSUB\RETURN können Programme in einfache Moduln unterteilt werden. Jedes Unterprogramm stellt eine Aufgabe dar. Die sinnvolle Zusammenfassung einzelner Aufgaben ist das Geheimnis effizienter Programmierung. Es ist nicht wichtig, wie komplex ein Programm ist, sondern wie einfach und gut es aufgebaut ist. Wird dies berücksichtigt, so können auch größere Aufgaben problemlos gelöst werden.

Dieses Kapitel kann am besten unter "Einfach gestalten" zusammengefaßt werden. Gliedern Sie ein Programm in die einzelnen Bestandteile auf, dann kann es wesentlich einfacher geschrieben werden.

Lektion 9

Bildschirmaufbau

Das Erstellen von Informationen mit einem Computer ist aufregend und lohnend. Die Art und Weise, in der die Informationen dargestellt werden, ist jedoch häufig genauso wichtig wie die Informationen selbst. Ebenso wie eine sauber aufgebaute und ausgedruckte Seite mehr Informationen übermittelt, als ein Bündel von Notizen auf einem Zettel, hat eine gut angeordnete Bildschirmanzeige eine wesentlich größere Auswirkung als eine Reihe von unordentlichen Zeichen auf dem Bildschirm.

Durch eine klare Bildschirmpräsentation werden nicht nur Ideen vermittelt, sondern kann auch das Programm gut aufgebaut werden. Denken Sie daran, wie Daten auf dem Bildschirm aussehen, so entscheiden Sie gleichzeitig, welche Folge das Programm einhalten muß, um die gewünschten Ergebnisse zu erhalten. Viele Programmierer entscheiden über das Erscheinungsbild der Bildschirme noch bevor sie mit dem Schreiben des Programms beginnen.

In diesem Abschnitt werden einige der Techniken erläutert, mit denen gute Bildschirmdarstellungen erstellt werden können. Hier wird beschrieben, wie Texte angeordnet, wichtige Wörter hervorgehoben und Menüs erstellt werden. Mit HTAB und VTAB kann der Text an eine beliebige Stelle auf dem Bildschirm gesetzt werden. Mit INVERSE können Großbuchstaben in dunklen Zeichen auf einem hellen Hintergrund (entgegen der sonst üblichen Anzeige) angezeigt werden. Durch NORMAL wird INVERSE ausgeschaltet. Sie werden sehen, wie die Anordnung von INPUT Bedienereingaben kontrolliert wird. Weiter werden Sie einen Algorithmus für das Zentrieren von Text kennenlernen.

Horizontale und vertikale Tabs

Bei einer Schreibmaschine werden die Tab-Stopps über die Seite verteilt. Bei Ihrem Computer wird mit HTAB festgelegt, wo der nächste Tab-Stopp sein soll.

Geben Sie das folgende Programm ein und führen Sie es aus:

```
NEW
10 HOME
20 HTAB 20
30 PRINT "HIER BIN ICH"
```

Ohne Zeile 20 wird die Meldung in der oberen linken Ecke des Bildschirms angezeigt. Mit der HTAB-Instruktion beginnt der Text 20 Zeichen weiter rechts. Der Bereich von HTAB geht von 0 bis 255. Mit dieser Instruktion wird der Text an eine beliebige Stelle auf dem Bildschirm gesetzt. Bei der 40-Zeichen-Anzeige wird jedes über Spalte 40 hinausgehende Zeichen des Textes auf die nächste Zeile gesetzt. So setzt HTAB 120 den Text beispielsweise drei Zeilen (120/40 = 3) tiefer.

Geben Sie das folgende Programm ein und führen Sie es aus:

```
10 HOME
20 INPUT "HTAB Wert (0-255) "; HZ
30 HTAB HZ
40 PRINT "X"
50 PRINT
60 HTAB 20
70 INPUT "Weitere HTAB? (J/N) "; AN$
80 IF AN$ = "J" THEN 10
90 PRINT "Danke fuer den Versuch!"      Der Benutzer weiß, das Programm ist beendet.
100 END
```

Normalerweise wird HTAB nur für die horizontale Anordnung des Textes benutzt. Für vertikale Tab-Stopps wird VTAB benutzt. VTAB wird wie HTAB benutzt, kann jedoch nur Werte von 1 bis 24 annehmen.

Um eine Vorstellung der Arbeitsweise von VTAB zu erhalten, führen Sie das nächste kleine Programm aus:

```
NEW
10 HOME
20 VTAB 10
30 PRINT "HIER BIN ICH"
```

Durch eine Kombination von HTAB und VTAB kann der Text an eine beliebige Stelle auf dem Bildschirm gesetzt werden.

Mit dem nächsten Programm können Angaben an eine beliebige Stelle auf dem Bildschirm gesetzt werden. Geben Sie das Programm ein und führen Sie es aus:

```
10 HOME
20 INPUT "HTAB Position (1 - 40)  "; HZ
25 IF HZ > 40 THEN PRINT "Zu gross" : GOTO 20
30 INPUT "VTAB Position (1 - 24) "; VT
35 IF VT > 24 THEN PRINT "Zu gross!" : GOTO 30
40 HOME
50 VTAB VT : HTAB HZ : PRINT "X"
60 VTAB 22 : HTAB 20
70 INPUT "Weiter? (J/N) "; AN$
80 IF AN$ <> "N" THEN 10
90 PRINT "Auf Wiedersehen."
100 END
```

In den Zeilen 50 und 60 stehen die HTAB- und VTAB-Instruktionen in derselben Zeile. Werden HTAB und VTAB auf diese Weise nacheinander gesetzt, so kann der Text etwas einfacher positioniert werden.

Mit HTAB und VTAB können stilvolle Programm-Menüs problemlos erstellt werden. Das nächste Programm benutzt eine FOR\NEXT Schleife, um Positionen für den Text zu generieren.

```
NEW
10 HOME
20 FOR X=1 TO 6
30 HTAB 10 : VTAB (2 * X)
35 REM Koennen Sie sich vorstellen, was Zeile 30 bewirkt?
40 GOSUB 100
50 PRINT X; ".  "; MENU$
60 NEXT X
70 VTAB 20 : HTAB 5
80 INPUT "Bitte Nummer eingeben: "; NUMMER
90 END
100 REM *********************
110 REM MENUE-AUSWAHL
120 REM *********************
130 IF X = 1 THEN MENU$ = "Hol den Hund rein"
140 IF X = 2 THEN MENU$ = "Schick die Katze raus"
150 IF X = 3 THEN MENU$ = "Spiel mit den Affen"
160 IF X = 4 THEN MENU$ = "Wisch den Boden auf"
170 IF X = 5 THEN MENU$ = "Lob den Computer"
180 IF X = 6 THEN MENU$ = "ENDE"
190 RETURN
```

Dieses Menü zeigt Ihnen nur, wie HTAB und VTAB benutzt werden. Dieses Konzept kann jedoch als Modell für Ihre eigenen Programme benutzt werden.

❖ *Warum Menüs mit Nummern?* In guten Menüs können die Benutzer ihre Auswahl mit einem oder zwei Tastenanschlägen treffen. Anhand des folgenden Beispielmenüs können Sie sehen, wie wichtig ein guter Menü-Aufbau ist:

```
Über welches Tier wünschen Sie Informationen?
Pachyderm
Pterodaktylus
Moorhuhn
Kreuzotter
Krokodil
Programm beenden
Bitte hier die Auswahl eintippen:
```

Dieses Menü garantiert praktisch von Anfang an Schreibfehler. Das Programm muß alle Arten von Fehlererkennungen umfassen, um die Schreibweise des Benutzers zu überprüfen. Mit numerierten Menüs wird das Problem umgangen:

```
Über welches Tier wünschen Sie Informationen?
1) Pachyderm
2) Pterodaktylus
3) Moorhuhn
4) Kreuzotter
5) Krokodil
6) Programm beenden
Bitte hier die Auswahl eingeben: (1 - 6):
```

Bei diesem Menü braucht der Benutzer nur eine Zahl einzugeben (und das Programm muß nur auf Einhaltung des Zahlenbereichs achten). Durch numerierte Menüs werden die Dinge sowohl für den Benutzer (der die Auswahl eingeben - und unter Umständen erneut eingeben - muß) als auch für den Programmierer (der das Programm schreiben muß) einfacher.

Anordnung der Bedienerhinweise

Bei einem guten Bildschirmaufbau muß darauf geachtet werden, wo die INPUT-Bedienerhinweise stehen. Programmierer müssen die Benutzer häufig zur Eingabe mehrerer Angaben in einer Zeile auffordern - mehrere Adressen, eine Reihe von Preisen oder Namen usw.

Geben Sie dieses Programm ein und führen Sie es aus. Es fragt nach Eingaben in schöner Form. Es werden HTAB und VTAB und ein neuer Programmier-Trick benutzt.

```
NEW
10 GOSUB 200
20 HOME
30 INPUT "Wie viele Namen werden es ? "; NAMEN
40 HOME
50 HTAB 5 : VTAB 10
60 PRINT "Die Namen nacheinander eingeben."
70 FOR X = 1 TO NAMEN
80 HTAB 17 : VTAB 10 : PRINT LEER$
90 HTAB 17 : VTAB 10
110 INPUT "Name: "; NA$
110 NEXT X
120 END
200 REM ************************
210 REM LEERMACHER
220 REM ************************
230 FOR S = 1 TO 20 : REM LEER$ ENTSPRICHT 20 STELLEN
240 LEER$ = LEER$ + " "
250 NEXT S
260 RETURN
```

Darauf kommen wir noch zurück.

Dies ist nur ein Beispiel. In einem "echten" Programm würden Sie nicht einfach Namen anfordern und sie gleich wieder wegewerfen!

Der Leermacher: Mit dem Unterprogramm in Zeile 200 wird ein praktischer Programmiertrick vorgestellt. Sie hätten LEER$ folgendermaßen definieren können:

```
LEER$ = "                    "
```

20 Leerstellen.

Dadurch erhalten Sie jedoch keine gute Vorstellung davon, wie viele Leerstellen zwischen den Anführungszeichen stehen. Wird eine Schleife für das Erstellen von LEER$ benutzt, wie im Unterprogramm in Zeile 200, so können Sie genau sehen, wie groß die "leere" Variable ist.

Natürlich können nicht nur Leerzeichen benutzt werden. Statt Leerzeichen, benutzen Sie Bindestriche oder Unterstreichungszeichen. Der Kreativität sind keine Grenzen gesetzt - ändern Sie einfach die Angaben in den Anführungszeichen in Zeile 240.

Hervorhebungen: INVERSE und NORMAL

Der Computer kann inverse Zeichen auf dem Bildschirm drucken. Mit der INVERSE-Instruktion wird der Text von hellen Zeichen auf dunklem Hintergrund in dunkle Zeichen auf hellem Hintergrund geändert. Der gesamte Text hinter einer INVERSE-Instruktion bleibt invers, bis das Programm eine NORMAL-Instruktion antrifft.

Geben Sie das folgende kleine Programm als Beispiel ein:

```
NEW
10 HOME
20 INVERSE
30 PRINT "DIES IST INVERS"
40 NORMAL
50 PRINT "DIES IST NORMAL"
```

Nehmen Sie Zeile 40 heraus, so wird der ganze Text invers gezeigt. Da mit inversem Text etwas hervorgehoben werden soll, wird empfohlen, die NORMAL-Instruktion hinzuschreiben, sobald die inverse Darstellung beendet ist.

Um den Benutzer darauf hinzuweisen, daß sein Computer Eingaben erwartet, benutzen Sie eine Eingabe-Anforderung in inverser Darstellung. In einem Menü wird eine Anforderung in gleicher Weise von der eigentlichen Menüauswahl abgehoben:

```
10 HOME
20 TITEL$ = "MENUE"
30 PRINT TITEL$
40 FOR X = 1 TO 4
50 HTAB 3 : VTAB (2 * X) + 2                Jetzt alles klar?
60 PRINT "Auswahl-Nummer "; X
70 NEXT X
80 VTAB 20 : INVERSE                        Inverse Darstellung beginnt...
90 INPUT "BITTE AUSWAEHLEN: "; WAHL$
100 NORMAL                                  und wird wieder beendet.
110 ...
```

❖ *INVERSE NUR BEI GROSSBUCHSTABEN:* INVERSE kann mit Kleinbuchstaben nicht gut benutzt werden. Aus seltsamen technischen Gründen werden Kleinbuchstaben gelegentlich in andere Zeichen umgewandelt, wenn sie invers dargestellt werden. Experimentieren Sie der Sicherheit halber, bevor Sie inverse Kleinbuchstaben in Ihren Programmen benutzen.

Üben Sie etwas mit INVERSE. Versuchen Sie eine Zeile mit Leerzeichen invers darzustellen. Zeigen Sie Ihren Namen invers an - benutzen Sie inversen Text mit Sternchen, um einen Filmtitel zu erzeugen und Ihren Namen in hellen Buchstaben anzuzeigen.

Ein Algorithmus zur Zentrierung von Text

Wie Sie in der letzten Lektion gesehen haben, sind Algorithmen Formeln, mit denen verschiedene Aufgaben ausgeführt werden. Bei allen Tricks, die in diesen Lektionen vorgestellt wurden, handelt es sich in Wirklichkeit um Algorithmen, die in Computercode übersetzt wurden. Nachdem Sie mit den Programmen in Ihrem Apple experimentiert haben, haben Sie wahrscheinlich schon einige eigene Algorithmen entwickelt.

Ein Algorithmus für das Zentrieren von Text ist besonders bequem, insbesondere in einer Lektion über das Formatieren von Bildschirmanzeigen. Für diesen Algorithmus wird die LEN Instruktion benötigt. Mit LEN wird die Länge einer Zeichenfolge berechnet. Nachfolgend ein Beispiel:

```
NEW
10 A$ = "Birnen weg!"
20 PRINT LEN( A$)
RUN
11
```

Klammern sind obligatorisch (wie bei RND).

Birnen weg! umfaßt elf Zeichen (einschließlich Leerzeichen).

Wird der Text zentriert, so wird die Hälfte der Zeichen links neben die Mitte der Zeile und die andere Hälfte rechts neben die Mitte gesetzt.

Nachdem dieses Grundprinzip bekannt ist, überlegen Sie, wie der Computer dieses Problem wohl löst. Schreiben Sie das Programm, testen Sie es und lesen Sie die Lösung im nächsten Abschnitt.

Eine Lösung für das Zentrieren

Auch hier ist dies nur eine von mehreren möglichen Lösungen. Haben Sie eine andere Lösung gefunden, die einwandfrei ausgeführt werden kann, so ist Ihre genauso gültig.

Hier der Algorithmus:

1. Ermitteln Sie die Anzahl von Zeichen, die auf eine Zeile passen; bei Ihrem Apple entweder 40 oder 80 Zeichen. Nehmen Sie die zutreffende Anzahl von Zeichen.

2. Ermitteln Sie die Länge der Zeichenfolge mit LEN.
3. Subtrahieren Sie die Länge der Zeichenfolge von der Bildschirmbreite; dividieren Sie das Ergebnis durch 2. Das Ergebnis ist die gesuchte Position.
4. Mit HTAB wird an diese Position gegangen.

Im Computercode sieht dies folgendermaßen aus:

```
HTAB (WIDTH - LEN (LETTERS$)) / 2
```

All diese Klammern sind erforderlich.

Benutzen Sie diesen Algorithmus, um einen eingegebenen zu Text zentrieren. Nachfolgend ein Beispiel, in dem der Algorithmus in einem Unterprogramm steht. Umfaßt Ihre Bildschirmanzeige 80 Zeichen, so muß die 40 in Zeile 130 in 80 geändert werden:

```
NEW
10 HOME
20 INPUT "Beliebiges Wort eingeben: "; W$
30 GOSUB 100
40 INVERSE : VTAB 20
50 INPUT "EIN WEITERES WORT (J/N) "; AN$
60 NORMAL : IF AN$ = "J" THEN 10
70 END
100 REM *****************
110 REM ZENTRIERT TEXT
130 HTAB (40 - LEN ( W$) )/2
140 VTAB 8
150 PRINT W$
160 RETURN
```

Zusammenfassung und Überblick

In dieser Lektion wurde auf die Bedeutung übersichtlicher Bildschirmanzeigen hingewiesen und wurden Programm-Menüs beschrieben.

Mit HTAB und VTAB kann der Text an eine beliebige Stelle auf dem Bildschirm gesetzt werden. Durch das Positionieren des Textes kann verdeutlicht werden, was gesagt werden soll oder was das Programm als nächstes vom Benutzer erwartet.

Sie haben gelernt, daß mit den INVERSE- und NORMAL-Instruktionen Text auf dem Bildschirm hervorgehoben wird. Mit diesen Instruktionen kann das Programm einfacher benutzt werden,weil wichtige Dinge deutlicher werden.

Außerdem haben Sie gelernt, daß der Algorithmus der Kern des Programmierens ist. Wie bei allen anderen Aspekten des Programmierens, können Sie Ihre eigenen Algorithmen erstellen, indem Sie eine Aufgabe in eine Reihe einfacher Teile aufgliedern. Jeder Algorithmus wird seinerseits zu einem Block des Programms.

Lektion 10

Benutzerfreundliches Programmieren

Herzlichen Glückwunsch! Damit haben Sie die Einführung in Applesoft BASIC und in die Grundlagen der Programmierung fast beendet. Die meisten Grundlagen, die Sie in diesem Tutorium gelernt haben, sind traditioneller Natur (sofern eine Wissenschaft, die erst 45 Jahre alt ist, überhaupt eine Tradition haben kann). In dieser letzten Lektion werden Sie etwas über neuere Tendenzen erfahren, die erst seit dem Auftreten der Personal Computer entwickelt wurden. Ihr Ziel ist es, "humanere" Programme zu schreiben, sie so aufzubauen, daß jeder Computeranfänger sie schnell erlernen und problemlos einsetzen kann.

Außerdem wird darauf hingewiesen, daß die Programmierung wesentlich schneller erlernt werden kann, wenn einer Benutzergruppe beigetreten wird, Programmierkurse belegt, Bücher gelesen und Computermagazine abonniert werden.

Eine unschöne Vergangenheit

In den guten alten Zeit (d.h. vor 1980 oder so) haben Programmierer praktisch keine Zeit darauf verwendet, ihren Computern beizubringen, wie sie sich gegenüber Menschen verhalten müssen. Programmierer konzentrierten sich in erster Linie darauf, daß ihre Programme liefen, ohne allzu häufig von Fehlermeldungen unterbrochen zu werden. Da sie im allgemeinen die einzigen waren, die ihre Programme benutzen, brauchten ihre Programme nicht "benutzerfreundlich" zu sein. Dies war zu diesem Zeitpunkt durchaus in Ordnung. Die meisten Benutzer programmierten für sich selbst und teilten ihre Programme nicht mit vielen anderen Benutzern.

Seither hat es jedoch einschneidende Änderungen gegeben. Millionen von Menschen besitzen heute Computer und viele Tausende schreiben Programme für sich selbst, ihre Kollegen und Freunde. Die meisten Programmierer, sowohl Hobby-Programmierer als auch berufsmäßige Programmierer sind Mitglieder von Benutzergruppen - Vereinigungen von Computer-Besitzern, die monatlich (einige Fanatiker sogar wöchentlich) zusammenkommen, um ihre Erfahrungen, Entdeckungen und selbsterstellten Programme auszutauschen.

Sollten Sie sich einer dieser ständig wachsenden Gruppen von Programmierern anschließen (und wenn Sie weiter programmieren, werden Sie dies wahrscheinlich tun), so müssen Sie Ihre Programme so einfach wie möglich gestalten. Die Idee, daß Programme und Computer für Menschen und nicht umgekehrt gemacht werden, ist in vielen dieser Benutzerkreisen noch revolutionär. Aber dennoch setzt sich dieser Gedanke immer mehr durch.

Richtlinien für benutzerfreundliche Programme

Nachfolgend einige Prinzipien, die beim Schreiben von benutzerfreundlichen Programmen berücksichtigt werden sollten. Diese Liste ist bei weitem nicht erschöpfend, reicht jedoch für den Anfang aus:

Klare Bedienerhinweise. Damit die Benutzer problemlos sehen können, was das Programm erwartet, wenn Informationen angefordert werden, muß das Programm genau angeben, was es haben möchte. Bedienerhinweise müssen hervorgehoben werden, einfach im Wortlaut sein und eine Reihe von Auswahlmöglichkeiten bieten, sofern diese vorhanden sind.

Sichern gegen Fehleingaben. Menschen machen Fehler. Das Programm sollte so viele Fehler wie möglich entdecken und den Benutzern die Chance geben, den Fehler zu korrigieren. Das Programm kann problemlos eine Prüfung auf die beiden häufigsten Probleme vornehmen: Bereichsfehler und Tippfehler. Bei einem Bereichsfehler gibt der Benutzer etwas ein, das nicht innerhalb des vom Computer oder Programm zugelassenen Bereichs liegt:

```
110 INPUT "Ihre Auswahl - 0, 1, 2 oder 3: "; AUSWAHL
120 IF AUSWAHL < 4 THEN GOTO 170
125 REM Verzweigt, wenn die Auswahl in Ordnung ist.
130 PRINT "Bedaure - Auswahl muß 0, 1, 2 oder 3 sein."
135 REM Fehler wird abgefangen.
140 PRINT "Bitte andere Auswahl treffen."
150 PRINT
160 GOTO 110 Geht für einen weiteren Versuch zurück.
170 REM macht weiter, wenn OK.
```

Bei einem Tippfehler gibt der Benutzer etwas ein, das er nicht wirklich eingeben wollte, oder buchstabiert ganz einfach etwas falsch:

```
90 ...
100 INPUT "Name des zu loeschenden Programms: "; LOESCHEN$
110 PRINT
120 INVERSE
130 PRINT "WARNUNG! BEIM LOESCHEN VON "
135 REM Gibt eine Warnung ab.
140 PRINT LOESCHEN$
145 REM Eingabe wird neu ausgedruckt
150 PRINT " IST ES FUER IMMER WEG"
160 PRINT
170 NORMAL
180 INPUT "PROGRAMM LOESCHEN? (J//N) "; KILL$
190 IF KILL$ <> "J" THEN HOME: GOTO 100
195 REM Wird geloescht, wenn nicht erfuellt.
200 ...
```

Immer einen Weg offen lassen. Vergessen Sie nicht, den Benutzern immer einen Weg offen zu lassen, das Programm zu verlassen. So wunderbar Ihr Programm auch sein mag, die Benutzer möchten auch einmal andere Dinge tun, wie beispielsweise Essen, in die Schule gehen oder Urlaub machen. Es gibt verschiedene Möglichkeiten, mit denen festgelegt werden kann, wann der Benutzer das Programm nicht mehr benötigt.

So können Sie beispielsweise am Anfang fragen, wie viele Eingaben der Benutzer vornehmen will:

```
90 ...
100 INPUT "Wie viele Schecks sind ausgestellt? "; SCHECKS
110 FOR X = 1 TO SCHECKS
120 ...
```

Sie können auch nach jedem Eintrag eine Ende-Option vorsehen:

```
90 ...
100 INPUT "Wie hoch ist der Scheck? DM "; BETRAG
110 SALDO = SALDO - BETRAG
120 INPUT "Weiterer Scheck? (J/N): "; ANS$
130 IF ANS$ = "N" THEN GOSUB 1000
135 REM Saldo angeben und Ende
140 ...
```

Das Programm kann auch mit einer Ende-Option nach jedem Eintrag oder nach jeder Folge von Einträgen in ein Menü zurückkehren:

```
90 ...
100 PRINT "1. Weitere Namen eingeben"
110 PRINT "2. Eintrag aendern"
120 PRINT "3. Alle Eintraege ausdrucken"
130 PRINT "4. Programm beenden"
140 VTAB 22 : HTAB 25
150 INPUT "Ihre Auswahl: "; AUSWAHL
160 IF AUSWAHL = 4 THEN END
170 ...
```

Um die Reaktionen auf die einzelnen Maßnahmen zu sehen, geben Sie jedes der beiden folgenden Programme ein und führen Sie es aus; bei dem ersten Programm werden die Regeln nicht befolgt, während das zweite Programm sie befolgt:

Rücksichtslose Programmierung (Igitt!)

```
10 INPUT A
20 SUM=SUM + A
30 PRINT SUM
40 GOTO 10
```

Benutzerfreundliche Programmierung (Super!)

```
10 HOME
20 INPUT "Zu addierender Betrag (0 = Ende)"; BETRAG
25 REM Betrag wird angefordert.
30 IF BETRAG = 0 THEN GOTO 130
35 REM Ende, wenn Benutzer fertig.
40 PRINT
50 PRINT "Sie gaben "; BETRAG; " DM ein, soweit OK? (J/N)";
60 INPUT ""; JN$ : REM Eintrag in Ordnung?
70 IF JN$ = "N" THEN GOTO 20
75 REM Wenn nicht, noch einmal anfordern.
80 SUM = SUM + BETRAG : REM Laufenden Saldo rechnen...
90 PRINT
100 PRINT "Laufender Saldo ist "; SUM : REM und angeben.
110 PRINT
120 GOTO 20 : REM Eine weitere Zahl anfordern
130 PRINT
140 PRINT "Endsaldo: "; SUM : REM Endsaldo ausdrucken.
```

Benutzerfreundliches Programmieren ist gar nicht so einfach

Das zweite Programm erfordert mehr Aufwand als das erste. Es bedarf mehr Planung, mehr Eingabeaufwand und mehr Austesten, um ein gutes interaktives Programm zu schreiben (d.h. ein Programm, das gut mit dem Benutzer kommuniziert). Allerdings lohnt sich der Aufwand. Menschen machen Fehler; dies muß beim Schreiben von Programmen immer berücksichtigt werden.

Es wird einfacher

Je mehr Sie über die Programmierung lernen, desto einfacher wird sie. Während Sie bei Ausführung dieser Lektionen eine ganze Reihe von Dingen gelernt haben, können Sie doch noch wesentlich mehr mit weiteren Instruktionen, Befehlen und Funktionen tun. Nachdem Sie eine Weile programmiert haben, werden Sie feststellen, daß Sie für Programme, die früher 20 Zeilen belegten, nur noch 5 Zeilen brauchen. Durch Experimentieren, Spielen und Erproben neuer Dinge mit Ihrem Apple Computer wachsen Ihre Programmierfähigkeiten wesentlich schneller, als Sie es sich vorstellen können.

Wie geht es weiter?

Sollten Sie der Auffassung sein, daß das Programmieren nichts für Sie ist, so ist dies kein Problem. Sie brauchen nicht zu wissen, wie ein Verbrennungsmotor arbeitet, um ein Auto fahren zu können. Und Sie brauchen auch nicht programmieren zu können, um einen Computer zu benutzen. Wenn Ihnen dieses Tutorium jedoch Spaß gemacht hat und Sie festgestellt haben, daß Programmieren eine feine und interessante Sache ist, so können Sie sich auf verschiedene Art und Weise weiterbilden.

Lesen Sie Bücher über Applesoft-Programmierung: Über Applesoft wurden hunderte von Büchern geschrieben, von Tutorien bis zu höher entwickelten technischen Dokumenten. Jede gute Computer-Buchhandlung verfügt mindestens über einige Applesoft Bücher, die größeren haben Dutzende davon. Absolut unerläßlich ist das *Applesoft BASIC Programmer's Reference Manual*, das von Addison-Wesley (ISBN 0-201-17722-6) veröffentlicht wurde. Von Fachleuten bei Apple Computer geschrieben, ist dies das offizielle Applesoft-Buch. Dieses Buch können Sie bei Ihrem Apple Fachhändler oder bei Ihrer Buchhandlung erwerben oder bestellen.

Beitritt zu einer Apple Benutzergruppe: Mit ihren Mitgliedern mit unterschiedlichstem Wissensstand sind die Apple Benutzergruppen der beste Freund des Computer-Anfängers. Sobald ein Mitglied etwas lernt, gibt er es an die anderen weiter. Die meisten Clubs verfügen über Gruppen für Anfänger; insbesondere haben alle besondere Arbeitsgemeinschaften für Applesoft BASIC, sowie für andere Computersprachen. (Logo, Pascal, C und Forth sind die bekanntesten unter ihnen.) Diese Gruppen sind nicht nur praktisch, sondern machen auch Spaß. In Europa ist die Apple User Group Europe (A.U.G.E.) die größte Benutzergruppe.

❖ *Kostenlose Software!* Das Schreiben von Programmen wird immer noch am einfachsten gelernt, indem einem anderem über die Schulter geschaut wird. Sobald Sie einer Apple Benutzergruppe beitreten, haben Sie Zugriff zu kiloweise frei zugänglicher Software. Und viele dieser "public domain" Programme sind in Applesoft geschrieben.

Programmierkurse: Programmierkurse werden in Schulen, Universitäten, Volkshochschulen, Computerläden, Fachhochschulen und Benutzergruppen abgehalten. Fragen Sie den Lehrer nach dem Niveau des Kurses, bevor Sie sich für einen Kurs entscheiden. Wenn möglich, sprechen Sie mit einigen Kursteilnehmern. Danach können Sie sicher sein, daß der Kurs Ihren Anforderungen entspricht.

Abonnieren Sie Apple Computer-Magazine: Es gibt Dutzende von Computer-Magazinen, von denen sich viele auf Apple Computer spezialisiert haben. Stellen Sie fest, ob Sie ein Magazin finden können, das sich ausschließlich auf Ihr Apple-Modell spezialisiert hat. Einige Apple-Magazine befassen sich sowohl mit den Macintosh Computern als auch den Computern der Apple II Familie, während andere sich nur mit dem einen oder anderen Computer befassen. Wieder andere Magazine sind mehr für Benutzer als für Programmierer gedacht. Auch hier kann wieder die Benutzergruppe helfen. Nicht nur können Mitglieder Magazine empfehlen, die Spalten für Anfänger haben, sondern viele Clubs haben auch Bibliotheken mit alten Ausgaben, die Sie benutzen können.

An die Arbeit!

Programmieren wird immer noch am einfachsten in der Praxis erlernt! Schreiben Sie sinnlose und ernsthafte Programme, lange und kurze Programme, komplexe und einfache Programme. Nur programmieren Sie! Sie lernen mehr aus einer Stunde, in der Sie sich mit Fehlern befassen, als in einem eine Woche dauernden Kurs. Schreiben Sie Programme nach Herzenslust.

Ein letztes Wort

Dieses kurze Handbuch sollte einige der wichtigsten Grundlagen bei der elementaren Programmierung vermitteln. Sie haben nicht alle Instruktionen in Applesoft BASIC kennengelernt. Es gibt viel zu viele, um sie in einem so kleinen Handbuch zu beschreiben. Was Sie jedoch hier gelernt haben, wird Ihnen beim Schreiben von Programmen helfen, wenn Sie nur daran denken, daß Sie auch für andere Benutzer schreiben.

Und immer schön weiter programmieren !

Anhang A

Zusammenfassung der Applesoft Instruktionen

Dies ist eine kurze Zusammenfassung sämtlicher Instruktionen in der Applesoft BASIC Sprache. Diese Zusammenfassung ist für die Programmierer gedacht, die schon Erfahrung mit anderen Computersprachen haben, jedoch noch nie mit Applesoft BASIC gearbeitet haben.

Für eine genaue Beschreibung dieser Instruktionen wird auf das *Applesoft BASIC Programmer's Reference Manual* (Addison-Wesley Publishing Company, Inc.) verwiesen.

ABS

```
ABS (-2.77)
```

Ermittelt den absoluten Wert (Wert ohne Berücksichtigung des Vorzeichens) des Arguments. Das Beispiel ergibt 2.77.

ASC

```
ASC ("QUEST")
```

Ergibt den ASCII-Code für das erste Zeichen in dem Argument. Dieses Beispiel ergibt 81 (ASCII-Code für Q).

ATN

```
ATN (0.8771)
```

Ergibt den Arcustangens des Argumentes in Radiant. Dieses Beispiel ergibt .720001187 (Radiant).

CALL

```
CALL -922
```

Führt ein Unterprogramm in Maschinensprache bei der in Dezimalform angegebenen Speicheradresse aus. Bei diesem Beispiel wird ein Zeilenvorschub ausgeführt.

CHR$

```
CHR$ (65)
```

Ergibt das Zeichen, das dem ASCII-Code entspricht, der als Argument angegeben wurde. Dieses Beispiel ergibt den Buchstaben A.

CLEAR

```
CLEAR
```

Setzt sämtliche Variablen und internen Steuerinformationen auf den Ausgangswert zurück. Der Programmcode bleibt unbetroffen.

COLOR=

```
COLOR= 12
```

Mit diesem Befehl wird die Bildschirmfarbe für niedrigauflösende Grafik festgelegt. Bei diesem Beispiel wird die grüne Farbe für die Anzeige gewählt.

CONT

```
CONT
```

Nimmt die Ausführung des Programms wieder auf, nachdem das Programm mit STOP, END, CONTROL-C oder (gelegentlich) mit CONTROL-RESET angehalten wurde.

COS

```
COS (2)
```

Ergibt den Cosinus des Argumentes, der in Radiant ausgedrückt werden muß. Das Beispiel ergibt -0.416146836.

DATA

```
DATA HANS SCHMIDT, "CODE 32", 23.45, -6
```

Mit diesem Befehl wird eine Liste von Elementen für die READ-Instruktion erstellt. In diesem Beispiel ist das erste Element die Zeichenfolge HANS SCHMIDT, das zweite die Zeichenfolge "CODE 32" , das dritte die reelle Zahl 23.45 und das vierte die Ganzzahl -6.

DEF FN

```
DEF FN KUBUS (X) = X * X * X
```

Definiert eine neue Funktion zur Benutzung in dem Programm. Mit diesem Beispiel wird eine Funktion definiert, die die Kubikzahl des Argumentes ergibt.

DEL

```
DEL 23, 56
```

Löscht einen Bereich von aufeinanderfolgenden Zeilen aus dem Programm. Mit diesem Beispiel werden die Zeilen 23 bis 56 einschließlich gelöscht.

DIM

```
DIM MARK (50,3), NAME$ (50)
```

Definiert und weist Platz für eine oder mehrere Matrizen zu. Mit dem Beispiel wird eine zweidimensionale reelle Matrix MARK definiert, deren erster Index von 0 bis 50 und deren zweiter Index von 0 bis 3 variiert, sowie eine Zeichenfolgenmatrix NAME$ mit einem Index, der von 0 bis 50 variiert.

DRAW

```
DRAW 4 AT 50, 100
DRAW 4
```

Zeichnet eine Form bei einem angegebenen Punkt im hochauflösenden Grafik-Bildschirm anhand der aktuellen Form-Tabelle. Mit dem ersten Beispiel wird Form Nummer 4 gezeichnet, beginnend in Spalte 50, Zeile 100, in der aktuellen Farbe, Maßstab und Drehparametern. Im zweiten Beispiel wird Form 4 bei dem letzten Punkt gezeichnet, der mit HPLOT, DRAW oder XDRAW gezeichnet wurde.

END

```
END
```

Beendet die Ausführung des Programms und gibt die Kontrolle an den Benutzer zurück. Es wird keine Meldung angezeigt.

EXP

```
EXP (2)
```

Ergibt den mathematischen Exponenten des Argumentes (d.h. die Konstante e - 2.7182818 - in die mit dem Argument angegebene Potenz erhoben). Das Beispiel ergibt e im Quadrat oder 7.3890561.

FLASH

```
FLASH
```

Durch diesen Befehl blinkt der ganze auf dem Bildschirm mit aufeinanderfolgenden PRINT-Anweisungen angezeigte Text zwischen hellen Zeichen auf dunklem Hintergrund und dunklen Zeichen auf hellem Hintergrund. Dieser Befehl kann unter Umständen mit Kleinbuchstaben (und anderen Zeichen mit größeren ASCII-Codes als 95) nicht richtig benutzt werden, wenn der Computer im "aktiven-80-Zeichen" -Modus arbeitet.

FN

```
FN KUBUS (6)
```

Wendet eine bestimmte Funktion auf den Wert des Argumentenausdrucks an. Wird von der Definition für die Funktion KUBUS unter DEF FN ausgegangen, so ergibt dieses Beispiel den Wert 216.

FOR

```
FOR J = 1 TO 10
FOR MARK = 0 TO 100 STEP 5
FOR ZAHL = 20 TO -20 STEP -2
```

Markiert den Anfang einer Schleife, identifiziert die Indexvariable und weist der Variablen Anfangs- und Endwerte, sowie (wahlweise) die Menge zu, um die der Wert bei jedem Durchlauf der Schleife geändert werden muß. Das erste Beispiel ist eine Schleife , deren Indexvariable J sämtliche Werte von 1 bis 10 in Schritten von 1 annimmt. Das zweite Beispiel ist eine Schleife, deren Indexvariable MARK Werte von 0 bis 100 in Schritten von 5 annimmt. Mit dem dritten Beispiel wird eine Schleife begonnen, deren Indexvariable ZAHL Werte von 20 bis -20 in Schritten von -2 annimmt.

FRE

```
FRE (0)
```

Ermittelt die Menge des verbleibenden Speicherplatzes in Bytes, die noch für das Programm zur Verfügung steht. Gleichzeitig wird eine Speicherbereinigung (garbage collection) der Zeichenketten-Variablen vorgenommen. Das Argument wird ignoriert, es muß sich jedoch um einen gültigen Applesoft Ausdruck handeln.

GET

```
GET ANTWORT$
```

Akzeptiert ein einzelnes Zeichen von der Tastatur, ohne es auf dem Bildschirm anzuzeigen und ohne daß die Return-Taste gedrückt werden muß. Das Programm wartet, bis der Benutzer eine Taste drückt. In diesem Beispiel wird das eingegebene Zeichen der Variablen ANTWORT$ zugewiesen.

GOSUB

```
GOSUB 250
```

Führt ein Unterprogramm ab der angegebenen Zeilennummer aus (hier: 250).

GOTO

```
GOTO 400
```

Übergibt die Kontrolle an die angegebene Zeilennummer (in diesem Beispiel 400).

GR

```
GR
```

Wandelt die Anzeige in 40 Zeilen eines niedrigauflösenden Grafik-Bildschirms mit vier Textzeilen am unteren Ende um. Der Bildschirm wird gelöscht und zeigt einen dunklen Hintergrund, der Cursor wird an den Anfang der letzten Zeile bewegt, und die Farbe für die niedrigauflösende Anzeige wird auf Schwarz festgelegt.

HCOLOR=

```
HCOLOR= 1
```

Legt die Bildschirmfarbe für hochauflösende Grafiken fest. Mit diesem Beispiel wird die Bildschirmfarbe auf Grün festgelegt.

HGR

```
HGR
```

Wandelt die Bildschirmanzeige in 160 Zeilen mit hochauflösenden Grafiken und vier Textzeilen am unteren Ende um. Der Bildschirm wird schwarz und Seite 1 der hochauflösenden Grafik wird angezeigt. Der Inhalt der Textanzeige, die Position des Cursors und die Farbe der hochauflösenden Anzeige bleiben unbetroffen.

HGR2

```
HGR2
```

Wandelt die Bildschirmanzeige in hochauflösende Grafiken über den ganzen Bildschirm (192 Zeilen) ohne Text um. Der Bildschirm wird schwarz und Seite 2 der hochauflösenden Grafik wird angezeigt. Der Inhalt der Textanzeige, die Position des Cursors und die Farbe für die hochauflösende Anzeige bleiben unbetroffen.

HIMEM:

```
HIMEM: 32767
```

Legt die Adresse der höchsten Speicherposition für das Applesoft-Programm, einschließlich seiner Variablen, fest. Im Beispiel wird das Ende des Programm- und Variablenspeichers auf 32767 festgelegt. Mit diesem Befehl wird ein Speicherbereich für Daten, hochauflösende Grafik oder Programmteile in Maschinensprache geschützt.

HLIN

```
HLIN 10, 20 AT 30
```

Zeichnet eine horizontale Linie in niedrigauflösender Grafik in der aktuellen niedrigauflösenden Farbe. Im Beispiel wird in Zeile 30 von Spalte 10 bis Spalte 20 gezeichnet.

HOME

```
HOME
```

Löscht den ganzen Text aus dem Textfenster und bringt den Cursor in die obere linke Ecke des Bildschirmfensters.

HPLOT

```
HPLOT 75, 20
HPLOT 48, 115 TO 79, 84 TO 110, 115
HPLOT TO 270, 10
```

Zeichnet einen Punkt oder eine Linie auf dem hochauflösenden Grafik-Bildschirm in der aktuellen Farbe für den hochauflösenden Bildschirm. Im ersten Beispiel wird ein einzelner Punkt in Spalte 75 und Zeile 20 gezeichnet. Im zweiten Beispiel werden Linien von Spalte 48, Zeile 115 zu Spalte 79, Zeile 84 und danach zu Spalte 110, Zeile 115 gezeichnet.Im dritten Beispiel wird ab dem letzten mit HPLOT gezeichneten Punkt eine Linie zu Spalte 270, Zeile 10 in der Farbe des als letztes gezeichneten Punktes gezeichnet (dies ist nicht unbedingt die aktuelle Anzeigefarbe).

HTAB

```
HTAB 23
```

Mit diesem Befehl wird der Cursor in eine bestimmte Spalte der Textanzeige gesetzt. Hier wird der Cursor in Spalte 23 bewegt.

IF...THEN

```
IF ALTER < 18 THEN A = 0 : B = 1: C = 2
IF ANTWORT$ = "JA" THEN GOTO 100
IF N > MAX THEN GOTO 25
IF N > MAX THEN 25
IF N > MAX GOTO 25
```

Mit diesem Befehl werden eine oder mehrere Instruktionen ausgeführt oder übersprungen, je nachdem, ob eine angegebene Bedingung wahr ist oder nicht. Im ersten Beispiel wird A auf 0, B auf 1 und C auf 2 gesetzt, wenn der Wert von ALTER kleiner ist als 18. Im zweiten Beispiel wird eine Verzweigung in Zeile 100 vorgenommen, wenn der Wert von ANTWORT$ die Zeichenfolge "JA" darstellt. Die drei letzten Beispiele verzweigen alle in Zeile 25, wenn der Wert von N größer ist als der Wert von MAX. Ist die angegebene Bedingung falsch, so wird die Ausführung in allen Fällen mit der nächsten Programmzeile fortgesetzt.

IN#

```
IN# 2
```

Gibt die Quelle für die nächste Eingabe an. Bei diesem Beispiel wird die nachfolgende Eingabe aus der Einheit in Steckplatz 2 gelesen.

INPUT

```
INPUT A%
INPUT "ALTER, EIN KOMMA, DANN NAMEN EINGEBEN"; ALTER, NAME$
```

Mit diesem Befehl wird eine Eingabezeile aus der aktuellen Eingabeeinheit gelesen. Bei dem ersten Beispiel wird ein Wert in die Variable A% gelesen. Bei dem zweiten Beispiel wird ein Bedienerhinweis angezeigt und werden danach die Werte in die Variablen ALTER und NAME$ gelesen.

INT

```
INT (98.6)
INT (-273, 16)
```

Ergibt den ganzzahligen Teil des Argument-Wertes. Die Beispiele ergeben 98 bzw. -274.

INVERSE

```
INVERSE
```

Mit diesem Befehl werden sämtliche mit nachfolgenden PRINT-Instruktionen ausgegebenen Großbuchstaben als dunkle Zeichen auf hellem Hintergrund und nicht als helle Zeichen auf dunklem Hintergrund angezeigt. Bei Kleinbuchstaben führt dieser Befehl zu nichtvorhersehbaren Ergebnissen.

LEFT$

```
LEFT$ ("APPLESOFT", 5)
```

Gibt eine bestimmte Anzahl von Zeichen ab dem Anfang der Zeichenfolge an. Dieses Beispiel führt zu der Zeichenfolge APPLE.

LEN

```
LEN ("NIEMALS EINE DUNKLE STUNDE")
```

Hiermit wird die Länge einer Zeichenfolge in Zeichen ermittelt. Das Beispiel ergibt 26.

LET

Siehe "Zuweisungs-Instruktion".

LIST

```
LIST
LIST 150
LIST 200-300
LIST 200, 300
```

Mit diesem Befehl wird das ganze oder ein Teil des Programms auf dem Bildschirm angezeigt oder in die aktuelle Ausgabeeinheit geschrieben. Mit dem ersten Beispiel wird das ganze Programm aufgelistet. Mit dem zweiten Beispiel wird nur Zeile 150 aufgelistet und mit den beiden letzten Beispielen die Zeilen 200 bis 300 einschließlich.

LOAD

```
LOAD DEMO
```

Mit diesem Befehl wird ein Programm von einer Diskette in den Speicher eingelesen. Im Beispiel wird ein Programm von einer Diskettendatei namens DEMO gelesen.

LOG

```
LOG (2)
```

Mit diesem Befehl wird der natürliche Logarithmus des Argumentes ermittelt. Dieses Beispiel ergibt einen Wert von .693147181.

LOMEM:

```
LOMEM: 24576
```

Mit diesem Befehl wird die Adresse der niedrigsten Speicherposition angegeben, die dem Programm zur Speicherung von Variablen zur Verfügung steht. Mit dem Beispiel wird der Anfang der Variablenspeicherung auf 24576 festgelegt.

MID$

```
MID$ ("JEDEN TAG EIN APFEL", 7, 3)
MID$ ("JEDEN TAG EIN APFEL", 7)
```

Mit diesem Befehl wird eine Anzahl von Zeichen ab der angegebenen Position in einer bestimmten Zeichenfolge ausgegeben. Bei dem ersten Beispiel wird die Zeichenfolge TAG und bei dem zweiten Beispiel TAG EIN APFEL angegeben.

NEW

```
NEW
```

Mit diesem Befehl wird das aktuelle Programm aus dem Speicher gelöscht und sämtliche Variablen und internen Kontrollinformationen in den Ausgangsstatus zurückgesetzt.

NEXT

```
NEXT
NEXT INDEX
NEXT J, I
```

Mit diesem Befehl wird das Ende einer Schleife angegeben und die Schleife für den nächsten Wert der Indexvariablen wiederholt, wie in der entsprechenden FOR-Instruktion angegeben. Mit dem ersten Beispiel wird die als letzte eingegebene Schleife beendet. Bei dem zweiten Beispiel wird die Schleife beendet, deren Indexvariable INDEX lautet. Bei dem dritten Beispiel wird das Paar mit verschachtelten Schleifen beendet, deren Indexvariablen J und I lauten.

NORMAL

```
NORMAL
```

Mit diesem Befehl wird der ganze aufgrund nachfolgender PRINT-Instruktionen auf dem Bildschirm angezeigte Text wie üblich mit hellen Zeichen auf dunklem Hintergrund angezeigt. Mit diesem Befehl wird INVERSE rückgängig gemacht.

NOTRACE

```
NOTRACE
```

Mit diesem Befehl wird die Anzeige von Zeilennummern für jede ausgeführte Instruktion gestoppt. Er macht TRACE rückgängig.

ON...GOSUB

```
ON ID GOSUB 100, 200, 23, 4005, 500
```

Wählt ein Unterprogramm zur Ausführung je nach Wert eines Ausdrucks. Bei diesem Beispiel wird die Kontrolle an ein Unterprogramm in Zeile 100, 200, 23, 4005 oder 500 übertragen, je nachdem, ob der Wert von ID gleich 1, 2, 3, 4 oder 5 ist. Hat ID keinen dieser Werte, so wird die Ausführung mit der nächsten Instruktion fortgesetzt.

ON...GOTO

```
ON ID GOTO 100, 200, 23, 4005, 500
```

Hiermit wird eine Zeilennummer ausgewählt, zu der, je nach Wert des Ausdrucks, die Verzweigung vorgenommen wird. Im Beispiel wird die Kontrolle an Zeile 100, 200, 23, 4005 oder 500 übertragen, je nachdem, ob ID gleich 1, 2, 3, 4 oder 5 ist. Hat ID keinen dieser Werte, wird die Ausführung mit der nächsten Instruktion fortgesetzt.

ONERR GOTO

```
ONERR GOTO 500
```

Mit diesem Befehl wird der normale Mechanismus zur Fehlerbehandlung bei Applesoft durch ein Unterprogramm ersetzt, das bei einer bestimmten Zeilennummer beginnt. Bei diesem Beispiel wird ein Unterprogramm zur Fehlerbehandlung in Zeile 500 gelegt.

PDL

```
PDL (1)
```

Liest die aktuelle Einstellung bei einem bestimmten Handregler. Mit diesem Beispiel wird die Einstellung von Handregler 1 gelesen.

PEEK

```
PEEK (37)
```

Dieser Befehl ermittelt den Inhalt einer bestimmten Speicherposition. Mit diesem Beispiel wird der Inhalt von Speicherposition 37 gelesen, in der die aktuelle Vertikalposition des Textcursors auf dem Bildschirm steht.

PLOT

```
PLOT 10, 20
```

Zeichnet einen einzelnen Block mit der aktuellen Anzeigefarbe bei einer angegebenen Position auf dem niedrigauflösenden Grafik-Bildschirm. Mit diesem Beispiel wird ein Block in Spalte 10, Zeile 20 gezeichnet.

POKE

```
POKE -16302, 0
```

Speichert einen Wert in eine bestimmte Speicheradresse. Mit diesem Beispiel wird der Wert 0 in Speicherposition 49234 (65536-16302) gespeichert, wodurch der Bildschirm von Grafik und Text gemischt in Grafik über den vollen Bildschirm umgeschaltet wird.

POP

```
POP
```

Mit diesem Befehl wird die letzte Rücksprungadresse vom Kontroll-Stack gelöscht, so daß die nächste RETURN-Instruktion die Kontrolle an die Instruktion übergibt, die auf die zweite zuletzt ausgeführte GOSUB-Instruktion folgt.

POS

```
POS (0)
```

Mit diesem Befehl wird die aktuelle Horizontalposition des Cursors auf dem Text-Bildschirm ermittelt. Das Argument wird ignoriert, allerdings muß es sich um einen gültigen Applesoft-Ausdruck handeln.

PR#

```
PR#1
```

Gibt den Bestimmungsort für die nächste Ausgabe an. Durch dieses Beispiel wird die nächste Ausgabe an die Einheit in Steckplatz 1 gesendet.

PRINT

```
PRINT
PRINT A$, "X = "; X
```

Schreibt eine Ausgabezeile in die aktuelle Ausgabeeinheit. Mit dem ersten Beispiel wird eine Leerzeile geschrieben. Mit dem zweiten Beispiel wird der Wert der Variablen A$ geschrieben, auf die bei der nächsten Tab-Position die Zeichenfolge "X =" folgt, auf die wiederum der Wert der Variablen X folgt.

READ

```
READ A, B%, C$
```

Liest Werte aus DATA-Instruktionen in den Hauptteil des Programms. Mit diesem Beispiel werden Werte in die Variablen A, B% und C$ gelesen.

REM

```
REM DIES IST EINE BEMERKUNG
```

Mit diesem Befehl werden Bemerkungen in den Hauptteil eines Programms eingebaut, damit es für den Benutzer verständlicher wird.

RESTORE

```
RESTORE
```

Führt dazu, daß die als nächstes ausgeführte READ-Instruktion bei dem ersten Element der ersten DATA-Instruktion in dem Programm mit dem Lesen beginnt.

RESUME

```
RESUME
```

Am Ende einer Routine zur Fehlerbehebung (siehe ONERR GOTO) wird das Programm am Anfang der Instruktion wieder aufgenommen, in der der Fehler aufgetreten ist.

RETURN

```
RETURN
```

Die letzte Instruktion des Unterprogramms gibt die Kontrolle an die Instruktion, die auf den GOSUB Befehl folgt, der das Unterprogramm aufgerufen hat.

RIGHT$

```
RIGHT$ ("APPLESOFT", 4)
```

Mit diesem Befehl wird eine bestimmte Anzahl von Zeichen vom Ende einer Zeichenfolge angegeben. Mit diesem Beispiel wird die Zeichenfolge "SOFT" angegeben.

RND

```
RND (1)
```

Dieser Befehl erzeugt eine Zufallszahl zwischen 0 und 1. Argumentenwerte von Null und negative Argumentenwerte führen zu wiederholten Folgen von Zufallszahlen.

ROT=

```
ROT= 16
```

Mit diesem Befehl wird die Winkeldrehung für hochauflösende Formen festgelegt, die mit DRAW oder XDRAW gezeichnet werden. Mit diesem Beispiel wird die Form um 90 Grad im Uhrzeigersinn gedreht.

RUN

```
RUN
RUN 500
RUN DEMO
```

Führt ein Applesoft Programm aus. Mit dem ersten Beispiel wird das gerade im Speicher stehende Programm ab dem Anfang ausgeführt. Mit dem zweiten Beispiel wird das Programm im Speicher ab Zeile 500 ausgeführt. Mit dem dritten Beispiel wird ein Programm aus einer Diskettendatei namens DEMO geladen und ausgeführt.

SAVE

```
SAVE DEMO
```

Mit diesem Befehl wird das gerade im Speicher stehende benannte Applesoft Programm auf eine Diskette geschrieben. Hier wird das Programm in eine Diskettendatei namens DEMO geschrieben.

SCALE=

```
SCALE= 10
```

Mit diesem Befehl wird der Skalierungsfaktor für hochauflösende Formen festgelegt, die mit DRAW oder XDRAW gezeichnet werden. Mit diesem Beispiel wird die Form zehnmal größer als mit der Definition in der Formtabelle angegeben gezeichnet.

SCRN

```
SCRN (10, 20)
```

Mit diesem Befehl wird der Code für die Farbe ermittelt, die gerade bei einer angegebenen Position in dem niedrigauflösenden Grafikbildschirm angezeigt wird. Dieses Beispiel ermittelt den Code für die Farbe in Spalte 10, Zeile 20.

SGN

```
SGN (-144)
```

Dieser Befehl ermittelt einen Wert von -1, 0 oder + 1, je nach Vorzeichen des Arguments. Dieses Beispiel ergibt -1.

SIN

```
SIN (2)
```

Dieser Befehl ermittelt den Sinus des Argumentes, der in Radiant ausgedrückt werden muß. Das Beispiel ergibt .909297427.

SPC

```
SPC (8)
```

Mit diesem Befehl wird eine Reihe von Leerzeichen in die Zeile eingefügt, die mit einer PRINT-Instruktion geschrieben wird. Hier werden acht Leerzeichen geschrieben.

SPEED=

```
SPEED= 50
```

Mit diesem Befehl wird die Geschwindigkeit festgelegt, mit der Textzeichen auf den Bildschirm oder an eine andere Ein-/Ausgabeeinheit gesendet werden. Die niedrigste Geschwindigkeit ist 0 und die höchste 255.

SQR

```
SQR (2)
```

Mit diesem Befehl wird die positive Quadratwurzel des Argumentes ermittelt; dieses Beispiel ergibt 1.41421356.

STOP

```
STOP
```

Mit diesem Befehl wird die Ausführung des Programms beendet und die Kontrolle an den Benutzer zurückgegeben. Mit einer Meldung wird die Programmzeile angegeben, in der die STOP-Instruktion steht.

STR$

```
STR$ (12.45)
```

Mit diesem Befehl wird eine Zeichenfolge ermittelt, die den numerischen Wert des Argumentes darstellt. Dieses Beispiel ergibt die Zeichenfolge "12.45".

TAB

```
TAB (23)
```

Dieser Befehl setzt den Textcursor während der Ausführung einer PRINT-Instruktion in eine bestimmte Position auf der Ausgabezeile. Hier wird der Cursor in Spalte 23 bewegt.

TAN

```
TAN (2)
```

Mit diesem Befehl wird der Tangens des Arguments ermittelt, das in Radiant angegeben wird. Dieses Beispiel ergibt -2.18503987.

TEXT

```
TEXT
```

Schaltet die Anzeige auf 24 Textzeilen um, der Cursor ist am Anfang der untersten Zeile steht.

TRACE

```
TRACE
```

Mit diesem Befehl wird die Zeilennummer jeder Instruktion auf dem Bildschirm angezeigt, während sie ausgeführt wird.

USR

```
USR (3)
```

Mit diesem Befehl wird ein vom Benutzer angegebenes Unterprogramm in Maschinensprache ausgeführt, wobei ihm ein bestimmtes Argument übergeben wird. Das Unterprogramm wird über eine JMP-Instruktion aufgerufen, die in den Adressen $0A bis $0C gespeichert ist. Bei diesem Beispiel wird als Argument der Wert 3 übergeben.

VAL

```
VAL ("-3.7E4")
```

Mit diesem Befehl wird der numerische Wert zurückgegeben, der von der als Argument angegebenen Zeichenfolge dargestellt wird. Dieses Beispiel ergibt -37000.

VLIN

```
VLIN 10, 20 AT 30
```

Dieser Befehl zeichnet eine vertikale Linie in niedrigauflösender Grafik mit der gerade benutzten Farbe für niedrigauflösende Grafiken. Mit diesem Beispiel wird eine Linie ab Zeile 10 bis 20 in Spalte 30 gezeichnet.

VTAB

```
VTAB 15
```

Dieser Befehl setzt den Cursor in eine bestimmte Zeile der Textanzeige. Mit dem Beispiel wird der Cursor in Zeile 15 bewegt.

WAIT

```
WAIT 49347, 15
WAIT 49347, 15, 12
```

Mit diesem Befehl wird die Ausführung des Programms suspendiert, bis ein bestimmtes Bitmuster in einer bestimmten Speicheradresse erscheint. Dieser Befehl wird im allgemeinen benutzt, um auf ein Statussignal von einem Peripheriegerät zu warten. Die zweiten und (wahlweise) dritten Argumente stellen Masken dar: mit dem zweiten Argument wird angegeben, welche Bits der angegebenen Adresse von Interesse sind, während das dritte Argument die Werte angibt, die in diesen Bits getestet werden müssen. Im ersten Beispiel wird die Ausführung eingestellt, bis ein 1-Bit in einer der vier niedrigwertigen Bitpositionen in Adresse 49347 erscheint. Im zweiten Beispiel wird auf ein 1-Bit in Position 0 oder 1 oder ein 0-Bit in Position 2 oder 3 gewartet.

XDRAW

```
XDRAW 4 AT 50, 100
XDRAW 4
```

Mit diesem Befehl wird eine Form aus der gerade im Speicher stehenden Formtabelle bei einem angegebenen Punkt auf dem hochauflösenden Grafik-Bildschirm gezeichnet. Jeder Punkt in der Form wird mit der Komplementärfarbe der gerade an diesem Punkt angezeigten Farbe gezeichnet. Mit diesem Befehl wird im allgemeinen eine schon gezeichnete Form gelöscht. Bei dem ersten Beispiel wird Form Nummer 4 ab Spalte 50, Zeile 100 mit den aktuellen Skalierungs- und Drehungsparametern gelöscht. Mit dem zweiten Beispiel wird Form 4 bei dem letzten Punkt gelöscht, der mit HPLOT, DRAW oder XDRAW gezeichnet wurde.

Zuweisungs-Instruktion

```
LET A = 23.5S67
A$ = "HUMBUG"
```

Weist den Wert des Ausdrucks hinter dem = der Variablen zu, die vor dem Gleichheitszeichen steht. LET ist wahlweise.

Anhang B

Reservierte Wörter

Tabelle B-1 enthält eine Liste der reservierten Applesoft Wörter. In den meisten Fällen können diese Zeichenfolgen nicht als Variablennamen oder in Variablennamen benutzt werden.

Das Ampersand (&) ist für den internen Gebrauch von Applesoft und für vom Benutzer gelieferte Routinen in der Maschinensprache reserviert.

XPLOT ist ein reserviertes Wort, das keiner aktuellen Applesoft Anweisung entspricht.

Einige reservierte Wörter werden von Applesoft nur in bestimmten Kontexten erkannt:

COLOR, HCOLOR, ROT, SCALE und SPEED werden nur dann als reservierte Wörter interpretiert, wenn das nächste Zeichen, das kein Leerzeichen darstellt, ein Gleichheitszeichen (=) ist. Dies ist bei COLOR und HCOLOR nur von geringer Bedeutung, da das eingefügte reservierte Wort OR deren Benutzung als Variablennamen ohnehin untersagt.

HIMEM und LOMEM werden nur dann als reservierte Wörter interpretiert, wenn es sich bei dem nächsten von einem Leerzeichen abweichenden Zeichen um einen Doppelpunkt (:) handelt.

IN und PR werden nur dann als reservierte Wörter interpretiert, wenn es sich bei dem nächsten von einem Leerzeichen abweichenden Zeichen um ein Nummernzeichen (#) handelt.

SCRN, SPC und TAB werden nur dann als reservierte Wörter interpretiert, wenn es sich bei dem nächsten von einem Leerzeichen abweichenden Zeichen um eine linke Klammer handelt (.

ATN wird nur dann als reserviertes Wort interpretiert, wenn zwischen dem T und dem N kein Leerzeichen steht. Steht zwischen T und N ein Leerzeichen, so wird statt dessen von dem reservierten Wort AT und nicht von ATN ausgegangen.

TO wird als reserviertes Wort behandelt, es sei denn, vor ihm steht ein A und zwischen dem T und O ist ein Leerzeichen vorhanden. In diesem Fall wird anstelle von TO das reservierte Wort AT interpretiert.

Selbst wenn Sie keine reservierten Wörter in den Variablennamen aufnehmen, können sie gelegentlich unerwartet auftauchen und zu Problemen führen. So wird beispielsweise die Anweisung

```
100 FOR A = LOFT OR LEFT TO 15
```

als

```
100 FOR A = LOF TO RLEFT TO 15
```

interpretiert und führt zu einem Syntaxfehler. Um eine richtige Interpretation zu gewährleisten, werden Klammern benutzt:

```
100 FOR A = (LOFT) OR (LEFT) TO 15
```

Tabelle B-1. Reservierte Applesoft Wörter

$	FN	LEFT$	POP	SQR
ABS	FOR	LEN	POS	STEP
AND	FRE	LET	PR#	STOP
ASC		LIST	PRINT	STORE
AT	GET	LOAD		STR$
ATN	GOSUB	LOG	READ	
	GOTO	LOMEM:	RECALL	TAB(
CALL	GR		REM	TAN
CHR$		MID$	RESTORE	TEXT
CLEAR	HCOLOR=		RESUME	THEN
COLOR=	HGR	NEW	RETURN	TO
CONT	HGR2	NEXT	RIGHT$	TRACE
COS	HIMEM:	NORMAL	RND	
	HLIN	NOT	ROT=	USR
DATA	HOME	NOTRACE	RUN	
DEF	HPLOT			VAL
DEL	HTAB	ON	SAVE	VLIN
DIM		ONERR	SCALE=	VTAB
DRAW	IF	OR	SCRN(	
	IN#		SGN	WAIT
END	INPUT	PDL	SHLOAD	
EXP	INT	PEEK	SIN	XDRAW
	INVERSE	PLOT	SPC(	XPLOT
FLASH		POKE	SPEED=	

Glossar

Absturz: Stellt ein Programm unerwarteterweise den Betrieb ein und beschädigt oder zerstört dabei Informationen, so spricht man von einem Absturz.

Adresse: Eine Zahl, mit der etwas gekennzeichnet wird, wie beispielsweise eine Position im Speicher des Computers.

Algorithmus: Ein schrittweises Verfahren zur Lösung eines Problems oder zur Ausführung einer Aufgabe.

Anfangswert: Der Wert, der der Indexvariablen bei dem ersten Durchlauf einer Schleife zugewiesen ist.

Anzeige: Visuell dargestellte Informationen, insbesondere auf dem Bildschirm eines Video-Gerätes.

Anzeigen: Informationen visuell darstellen.

Apple II: Eine Familie mit Personal Computern, die von Apple Computer Inc. hergestellt und verkauft werden. Gattungsname für sämtliche Computer in dieser Serie.

Applesoft: Eine erweiterte Version der BASIC Programmiersprache, die mit der Apple II Computerfamilie benutzt wird und Zahlen in Gleitkommaform verarbeiten kann. Ein Interpreter für das Erstellen und Ausführen von Programmen in Applesoft ist in den ROM im Apple II System eingebaut.

Arithmetischer Operator: Ein Operator, wie beispielsweise +, mit dem numerische Werte kombiniert werden, um ein numerisches Ergebnis zu erzeugen. Siehe **Vergleichsoperator**.

Aufhängen: Dieser Ausdruck wird für ein Programm oder System benutzt, das endlos weiterarbeitet, ohne zu ernsthaften Ergebnissen zu führen.

Ausgabe: (1) Informationen, die von einem Computer an eine externe Einheit übertragen werden, wie beispielsweise den Bildschirm, ein Diskettenlaufwerk, einen Drucker oder ein Modem. (2) Die eigentliche Übertragung derartiger Informationen.

Ausführen: (1) Ausführung einer bestimmten Aktion oder Folge von Aktionen, wie beispielsweise vom Programm definiert. (2) Laden eines Programms von einem Datenträger, wie beispielsweise einer Diskette in den Hauptspeicher und Ausführen dieses Programms.

Austesten: Einen Fehler oder die Ursache eines Problems bzw. einer Fehlfunktion in einem Computersystem eingrenzen und beheben. Dieser Ausdruck wird typischerweise im Zusammenhang mit Software-bezogenen Problemen benutzt.

BASIC: **B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode ; eine Programmiersprache, die einfach zu erlernen und zu benutzen ist.

Bedienerhinweis: (1) Hinweis für den Benutzer, daß eine bestimmte Aktion erforderlich ist, im allgemeinen durch Anzeige eines bestimmten Symbols, einer Meldung oder eines Menüs mit Auswahlmöglichkeiten auf dem Bildschirm. (2) Eine Instruktion oder Meldung, die auf dem Bildschirm angezeigt wird.

Bedingter Sprung: Ein Sprung, der davon abhängt, ob eine Bedingung oder der Wert eines Ausdrucks wahr ist.

Befehl: Eine Anweisung vom Benutzer an ein Computersystem (im allgemeinen über die Tastatur), mit der der Computer zur Ausführung einer bestimmten Aktion angewiesen wird.

Benutzer: Die Person, die ein Computersystem bedient oder kontrolliert.

Benutzerschnittstelle: Die Regeln und Bestimmungen, mit denen ein Computersystem mit der Person kommuniziert, die es bedient.

Bildschirmeinheit: Eine Einheit, bei der Informationen visuell dargestellt werden, wie beispielsweise ein Fernsehempfänger oder ein Video-Monitor.

Bildschirm: Das Glas- oder Plastik-Feld auf der Vorderseite einer Bildschirmeinheit, auf dem Bilder angezeigt werden.

Bug: Ein Fehler in einem Programm, durch den das Programm nicht wie vorgesehen arbeitet.

Code: (1) Eine Zahl oder ein Symbol, mit der bzw. dem Informationen in einer kompakten oder einfach zu verarbeitenden Form dargestellt werden. (2) Die Anweisungen oder Instruktionen, aus denen ein Programm besteht.

Computer: Eine elektronische Einheit zur Ausführung vorher definierter (programmierter) Rechenoperationen mit hoher Geschwindigkeit und großer Genauigkeit.

Computersystem: Ein Computer und die zugehörige Hardware, Firmware und Software.

Cursor: Eine Marke oder ein Symbol, das auf dem Bildschirm angezeigt wird und angibt, wo die nächste Benutzeraktion stattfindet oder das nächste über die Tastatur eingegebene Zeichen angezeigt wird.

Datei: Eine Sammlung von Informationen, die als eine benannte Einheit auf einem Datenträger, wie beispielsweise einer Diskette, gespeichert werden.

Dateiname: Der Name, unter dem eine Datei auf einer Diskette gespeichert ist.

Definition: Zuweisung eines Wertes an eine Variable.

Diskette: Ein Speichermedium, das aus einer flachen kreisförmigen magnetischen Scheibe besteht, auf deren Oberfläche Informationen in Form von kleinen magnetisierten Punkten aufgezeichnet werden können, ähnlich wie Musik auf Magnetbändern aufgenommen wird.

Diskettenlaufwerk: Ein Peripheriegerät, mit dem Informationen auf der Oberfläche einer Diskette aufgezeichnet werden.

Drucker: Ein Peripheriegerät, mit dem Informationen in einer für den Menschen lesbaren Form auf Papier ausgedruckt werden.

Durchlauf: Eine einzige Ausführung einer Schleife.

Editieren: Ändern, z.B. Text in einem Dokument einfügen, löschen, ersetzen oder verschieben.

Eingabe: (1) Informationen, die von einer externen Quelle in einen Computer übertragen werden, wie beispielsweise der Tastatur, einem Diskettenlaufwerk oder Modem. (2) Die eigentliche Übertragung dieser Informationen.

Eingabevariable: Eine Variable, deren Wert über eine INPUT-Instruktion vom Benutzer zugewiesen wird, im Gegensatz zu einer Variablen, deren Wert vom Programmierer mit einer Zuweisungs- oder ähnlichen Instruktion zugewiesen wird.

Endlosschleife: Ein Programmabschnitt, der dieselbe Folge von Schritten endlos wiederholt.

Fehlermeldung: Eine auf dem Bildschirm angezeigte oder ausgedruckte Meldung, mit der der Benutzer über einen Fehler oder ein Problem bei der Ausführung eines Programms unterrichtet wird.

Firmware: Name für Programme, die im Nur-Lese-Speicher gespeichert sind.

Format: Die Form, in der Informationen erstellt oder dargestellt werden.

Formatieren: (1) Das Format von Informationen angeben oder steuern. (2) Vorbereitung einer leeren Diskette zur Aufnahme von Informationen, indem die Oberfläche in Spuren und Sektoren unterteilt wird. Wird auch als Initialisieren bezeichnet.

Grafik: (1) In Form von Bildern dargestellte Informationen. (2) Anzeige von Bildern auf dem Bildschirm eines Computers. Vergleiche **Text.**

Handregler: Ein Peripheriegerät, das an den Anschluß für Handregler des Apple II angeschlossen werden kann und über einen Drehregler und eine Taste verfügt. Der Handregler wird typischerweise für die Bedienung von Spielprogrammen benutzt, kann jedoch auch in ernsthafteren Anwendungen benutzt werden.

Hardcopy: Informationen, die für den menschlichen Gebrauch auf Papier ausgedruckt werden.

Indexvariable: Eine Variable, deren Wert sich bei jedem Schritt durch eine Schleife ändert; wird auch häufig als Kontrollvariable oder Schleifenvariable bezeichnet.

Information: Fakten, Konzepte oder Instruktionen, die in organisierter Form dargestellt werden.

Inhaltsverzeichnis: Eine Liste sämtlicher auf einer Diskette gespeicherten Dateien.

Initialisieren: (1) In einen Ausgangsstatus oder auf einen Ausgangswert setzen, um einen Rechenvorgang vorzubereiten. (2) Vorbereitung einer leeren Diskette zur Aufnahme von Informationen, indem die Oberfläche in Spuren und Sektoren unterteilt wird. Wird auch als Formatieren bezeichnet.

Instruktion: Eine Programmeinheit in einer höheren Programmiersprache, mit der ein vom Computer auszuführender Schritt angegeben wird, der normalerweise mehreren Instruktionen in der Maschinensprache entspricht.

Interaktiv: Arbeit mit Hilfe eines Dialogs zwischen dem Computersystem und dem Benutzer.

Interaktive Programmierung: Hierbei werden Programme generiert, die mit Hilfe eines Dialogs zwischen dem Computersystem und einem Benutzer arbeiten.

Inverse Darstellung: Die Anzeige des Textes auf dem Bildschirm eines Computers in Form von dunklen Punkten auf hellem Hintergrund (oder einem anderen farbigen Hintergrund) anstelle der üblichen hellen Punkte auf einem dunklen Hintergrund.

Kontrollvariable: Siehe Indexvariable.

Laden: Übertragung von Informationen von einem Datenträger (wie beispielsweise einer Diskette) in den Hauptspeicher zur Verarbeitung. Beispielsweise die Übertragung eines Programms in den Speicher zur Ausführung.

Lesen: Informationen von einer für den Computer externen Quelle (wie beispielsweise einem Diskettenlaufwerk oder dem Modem) in den Speicher des Computers oder von einer für den Prozessor externen Quelle (wie beispielsweise der Tastatur oder dem Hauptspeicher) in den Prozessor des Computers übertragen.

Menü: Eine Liste mit Auswahlmöglichkeiten, die im allgemeinen von einem Programm auf den Bildschirm angezeigt werden und aus der der Benutzer seine Wahl treffen kann.

Modus: (1) Eine von verschiedenen Möglichkeiten, mit denen ein Computer Informationen interpretiert. (2) Der Status eines Computers oder Systems, mit dem sein Verhalten bestimmt wird.

Niedrigauflösende Grafiken: Die Anzeige von Grafiken auf dem Bildschirm des Apple II in Form einer Matrix aus Blöcken mit 16 Farben, mit einer Breite von 40 Spalten und einer Höhe von 40 oder 48 Zeilen.

Numerische Variable: Siehe Variable.

Nur-Lese-Speicher: Ein Speicher, dessen Inhalt gelesen, aber nicht geschrieben werden kann. Er wird für die Speicherung der Firmware benutzt. Informationen werden nur einmal während der Herstellung in den Nur-Lese-Speicher geschrieben. Danach bleiben sie permanent dort, selbst wenn der Computer ausgeschaltet wird. Sie können niemals gelöscht oder geändert werden. Siehe auch **Lese-/Schreib-Speicher**.

Operator: Ein Symbol oder eine Folge von Zeichen, wie beispielsweise + oder AND, mit dem bzw. der eine Operation angegeben wird, die mit einem oder mehreren Werten (den Operanden) ausgeführt werden muß, um ein Ergebnis zu erhalten.

Priorität: Die Reihenfolge, in der Operatoren bei der Auswertung eines Ausdrucks benutzt werden.

Programm: Eine Gruppe von Instruktionen, mit der die von einem Computer für eine bestimmte Aufgabe auszuführenden Schritte beschrieben werden, wobei die Regeln und Bestimmungen einer bestimmten Programmiersprache eingehalten werden. Bei Applesoft eine Folge von Programmzeilen, jeweils mit einer anderen Zeilennummer.

Programmieren: Schreiben eines Programms.

Programmierer: Der Autor eines Programms, d.h. die Person, die die Programme schreibt.

Programmiersprache: Eine Gruppe von Regeln oder Bestimmungen für das Schreiben von Programmen.

Programmzeile: Die Grundeinheit eines Applesoft Programms, die aus einer oder mehreren Instruktionen besteht, die durch Doppelpunkte (:) voneinander getrennt werden.

RAM: Siehe **Schreib-/Lese-Speicher**.

Reserviertes Wort: Ein Wort oder eine Zeichenfolge, das bzw. die von einer Programmiersprache für eine bestimmte Benutzung reserviert wird und deshalb nicht als Variablenname in einem Programm benutzt werden kann.

ROM: Siehe **Nur-Lese-Speicher**.

Routine: Ein Teil eines Programms, mit dem eine Aufgabe ausgeführt wird, die von der Gesamtaufgabe des Programms bestimmt wird.

Schleife: Ein Abschnitt eines Programms, der wiederholt ausgeführt wird, bis eine bestimmte Bedingung eintritt, wie beispielsweise eine Indexvariable, die einen angegebenen Endwert erreicht.

Schleifenvariable: Siehe **Indexvariable**.

Schnittstelle: Einheiten, Regeln oder Konventionen, anhand derer eine Systemkomponente mit einer anderen kommuniziert.

Schreiben: Übertragung von Informationen von dem Computer an eine externe Einheit (wie beispielsweise ein Diskettenlaufwerk, einen Drucker oder ein Modem) oder von dem Prozessor des Computers an eine externe Einheit (wie beispielsweise den Hauptspeicher).

Schreib-/Lese-Speicher: Speicher, dessen Inhalt sowohl gelesen als auch beschrieben werden kann. Auf den Inhalt einer einzelnen Position im Schreib-/Lese-Speicher kann in willkürlicher oder beliebiger Reihenfolge Bezug genommen werden. Die in dieser Art von Speicher enthaltenen Informationen werden gelöscht, sobald der Computer ausgeschaltet wird. Sie sind für immer verloren, wenn sie nicht auf einem Datenträger, wie beispielsweise einer Diskette, gesichert wurden. Siehe **Nur-Lese-Speicher**.

Schrittweise Verbesserung: Eine Technik der Programmentwicklung, bei der grobe Abschnitte des Programms zuerst festgelegt und dann Schritt für Schritt ausgearbeitet werden, bis ein vollständiges Programm erstellt ist.

Schrittwert: Der Wert, um den die Indexvariable bei jedem Durchlauf einer Schleife geändert wird.

Sichern: Informationen aus dem Hauptspeicher für den späteren Gebrauch auf einen Datenträger übertragen.

Sofortige Ausführung: Die Ausführung einer Applesoft Programmzeile sofort nach ihrer Eingabe. Die sofortige Ausführung wird benutzt, wenn die Zeile ohne eine Zeilennummer eingegeben wird. Siehe auch **verzögerte Ausführung**.

Speicher: Eine Komponente eines Computersystems, mit der Informationen für den späteren Abruf gespeichert werden können; siehe Hauptspeicher, Schreib-/Lese-Speicher, Nur-Lese-Speicher.

Sprache: Siehe Programmiersprache.

Syntax: Die Regeln, mit denen die Struktur der Anweisungen oder Instruktionen in einer Programmiersprache festgelegt werden.

System: Eine koordinierte Sammlung von zusammengehörigen und zusammenarbeitenden Teilen, die so aufgebaut sind, daß sie eine bestimmte Funktion ausführen oder einen bestimmten Zweck erfüllen.

Systemanzeige: Ein Textzeichen, das auf dem Bildschirm angezeigt wird, um den Benutzer zur Ausführung eines bestimmten Schrittes aufzufordern. Kennzeichnet auch häufig das Programm oder die Systemkomponente, von der die Aufforderung ausgeht. So benutzt der Applesoft BASIC Interpreter beispielsweise die Systemanzeige].

Tastatur: Die Tasten, wie auf einer Schreibmaschinentastatur, mit denen Informationen für den Computer eingegeben werden.

Text: (1) Informationen, die in Form von Zeichen dargestellt werden, die der Mensch lesen kann. (2) Anzeige von Zeichen auf dem Apple II Bildschirm. Siehe **Grafik**.

Unterprogramm: Ein Teil eines Programms, das auf Anforderung aus einer beliebigen Stelle in dem Programm heraus ausgeführt werden kann und die Kontrolle nach Abschluß wieder an die anfordernde Stelle zurückgibt.

Variable: (1) Eine Position im Speicher des Computers, in der ein Wert gespeichert werden kann. (2) Das Symbol, mit dem diese Position in einem Programm dargestellt wird.

Vergleichsoperator: Ein Operator, wie beispielsweise >, mit dem numerische Werte verglichen werden, um ein logisches Ergebnis zu erhalten. Siehe arithmetischer Operator.

Verketten: Zwei oder mehr Zeichenfolgen werden in einer einzigen, längeren Zeichenfolge kombiniert, die sämtliche Zeichen der Originalzeichenfolgen enthält.

Verschachtelte Schleife: Eine Schleife innerhalb einer anderen Schleife, die wiederholt während jedem Schritt durch die übergeordnete Schleife ausgeführt wird.

Verschachtelter Unterprogramm-Aufruf: Aufruf eines Unterprogramms aus einem anderen Unterprogramm.

Verzögerte Ausführung: Bei der verzögerten Ausführung wird eine Applesoft Programmzeile zur Ausführung zu einem späteren Zeitpunkt als Bestandteil eines vollständigen Programms gesichert. Die Ausführung wird verzögert vorgenommen, wenn die Zeile mit einer Zeilennummer eingegeben wird. Siehe **sofortige Ausführung.**

Verzögerungsschleife: Eine Schleife, mit der die Ausführung eines Programms verlangsamt werden soll.

Verzweigen: Die Programmausführung an eine Zeile oder Instruktion weitergeben, bei der es sich nicht um die nächste folgerichtige Zeile oder Instruktion handelt.

Wert: Ein Informationselement, das in einer Variablen gespeichert werden kann, wie beispielsweise eine Zahl oder eine Zeichenfolge.

Zähler: Eine Variable, mit der die Wiederholung einer Schleife verfolgt wird. Zähler haben häufig die Form X = X + 1.

Zeichen: Ein Buchstabe, eine Ziffer, ein Satzzeichen oder ein anderes geschriebenes Symbol, mit dem Informationen in einer für den Menschen lesbaren Form ausgedruckt oder angezeigt werden.

Zeichenfolge: Ein Informationselement, das aus einer Folge von Textzeichen besteht.

Zeichenfolgenvariable: Siehe **Variable**.

Zeile: Siehe **Programmzeile**.

Zeilennummer: Eine Nummer, mit der eine Programmzeile in einem Applesoft Programm gekennzeichnet wird.

Zeilenumbruch: Die automatische Fortsetzung des Textes am Anfang der nächsten Zeile, beispielsweise auf dem Bildschirm oder einem Drucker.

Index

R

S

T

U

Das Apple CAP System

Dieses Apple Handbuch wurde geschrieben, editiert und gesetzt mit dem Apple CAP-System (Computer Aided Publishing) mit Hilfe des Apple Macintosh™ Plus und Microsoft® Word. Korrektur und Satz erfolgten auf dem Apple LaserWriter™ Plus. Die Programmiersprache des LaserWriter, POSTSCRIPT™, wurde von Adobe Systems Incorporated entwickelt.

Der Textschrifttyp ist ITC Garamond® (ein ladbarer Zeichensatz, der von Adobe Sytems vertrieben wird). Der Bildschirm-Zeichensatz ist ITC Avant Garde Gothic®. Hervorhebungen sind in ITC Zapf Dingbats®, Programmlisten in Apple Courier gesetzt, einem dicktengleichen Zeichensatz.

Apple Computer GmbH
Ingolstädter Straße 20
D-8000 München 45
Tel: 089/3 50 34-0
Telex 5213 261

D-030-1318-A